AD-A102 693 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
MICROCOMPUTER BASED SHIP-BOARD GUN CONTROL SYSTEM.(U)
MAR 81 A ERDOGAN
UNCLASSIFIED NL

1 OF 7
AD
A102693

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MICROCOMPUTER BASED
SHIP-BOARD GUN CONTROL SYSTEM

by

Ahmet Erdogan

March 1981

Thesis Advisor:	U. Kodres
Co-Advisors:	R. Panholzer
	R. Baird

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A102	3. RECIPIENT'S CATALOG NUMBER 693
4. TITLE (and Subtitle) Microcomputer Based Ship-board Gun Control System		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1981
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ahmet Erdogan		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE March 1981
		13. NUMBER OF PAGES 570
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Servomechanism	Microprocessor
Ship-board Gun Control System	ADC/DAC
Microcomputer	Interactive Display
Closed-Loop System	
Plasma Display	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This study was undertaken to design and implement a micro-computer based gun control and interactive display system which is suitable as a model of a shipboard Gun Fire Control System and Tactical-Situation display. The stand-alone system includes two plasma display scopes, a microcomputer, a cathode ray tube (CRT), an analog-to-digital, digital-to-analog (ADC/DAC) board and a servo unit. The scope of the effort includes, calculation of

#20 - ABSTRACT - (CONTINUED)

target information, prediction of target values, solution of anti-air warfare and surface fire control problems. The servo unit was connected to the computer through the ADC/DAC board. The use of the servo unit and the true-motion plotter emulates the shipboard weapon system environment. Of major interest was the integration of the hardware components and the software developed in this study into a control of analog servo unit and a graphical display system.

Microcomputer Based Ship-board Gun Control System

by

Ahmet Erdogan
Lieutenant, Turkish Navy
B.S., Naval Postgraduate School, 1979

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

and

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

March 1981

Author: A Erdogan

Approved by: Uno R. Kodres
Thesis Advisor

C Robert Baird.
Co-Advisor

R Panholzer
Co-Advisor

[illegible]
Chairman, Department of Computer Science

[illegible]
Chairman, Department of Electrical Engineering

William M Tolles
Dean of Science and Engineering

Dean of Information and Policy Sciences

ABSTRACT

This study was undertaken to design and implement a microcomputer based gun control and interactive display system which is suitable as a model of a shipboard Gun Fire Control System and Tactical-Situation display. The stand-alone system includes two plasma display scopes, a micro-computer, a cathode ray tube (CRT), an analog-to-digital, digital-to-analog (ADC/DAC) board and a servo unit. The scope of the effort includes, calculation of target information, prediction of target values, solution of anti-air warfare and surface fire control problems. The servo unit was connected to the computer through the ADC/DAC board. The use of the servo unit and the true-motion plotter emulates the shipboard weapon system environment. Of major interest was the integration of the hardware components and the software developed in this study into a control of analog servo unit and a graphical display system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

An examination of currently used Naval Weapon Systems reveals that the heart of the system, the computer, and the tactical display system, do not represent today's advanced technology. For correct and instant decisions, the display system must display the whole picture of the tactical situation and interact with the fire control computer system.

Because a significant portion of the cost of building or maintaining a warship is the electronics in its sensor and weapons systems, and because a great amount of time and personnel power is employed in performing simple but important tasks, microcomputers offer the potential to:

1. Reduce the hardware cost of digital systems.
2. Perform complex calculations at main and/or remote stations and thus relieve the computational congestion at larger central computing facilities.
3. Perform functions faster and more accurately than currently handled by watch personnel, thus reducing decision and response times and the manning requirements of watch sections. Examples of these functions are: the problem of manual tracking of radar contacts, the solution of Maneuvering Board problems, target selection and evaluation, the transmission of weapon orders from CIC to the Weapons System Central.

The purpose of this study is to design and partially implement a microcomputer based gun fire control system with an interactive display system suitable for a model of a shipboard gun fire control system. The basic control system consists of a microcomputer, a single board computer interfaced with an ADC/DAC board, pre-amplifier unit, SERVO amplifier unit, motor tachometer unit and an operational amplifier unit. The display system consists of 2 plasma display scopes, a plasma touch panel and CRT which are all interfaced to the microcomputer. The shipboard tactical environment is simulated by interfacing the system to potentiometers via the ADC board which emulates a sensor (radar). The system block diagram is shown in Figure 1.

The system is designed to support data reception, calculation of weapon orders, remote processing (manipulation and handling of data to build the local data bases), information display of graphical and alphanumerical data and a Man-Machine Interface. The environment emulated by the potentiometers provides a data subset of the sensor (e.g., radar, sonar). The subset of information is provided via the interfaced ADC/DAC to the Single Board Computer (SBC) which in turn is interfaced to the microcomputer. The information consists of the air target coordinate values.

It is the purpose of this thesis to demonstrate the design approach of an accurate and fast weapon system. This system was developed and implemented by using a general purpose microcomputer system with an interactive plasma display which provides the man-machine user interface. The weapons system

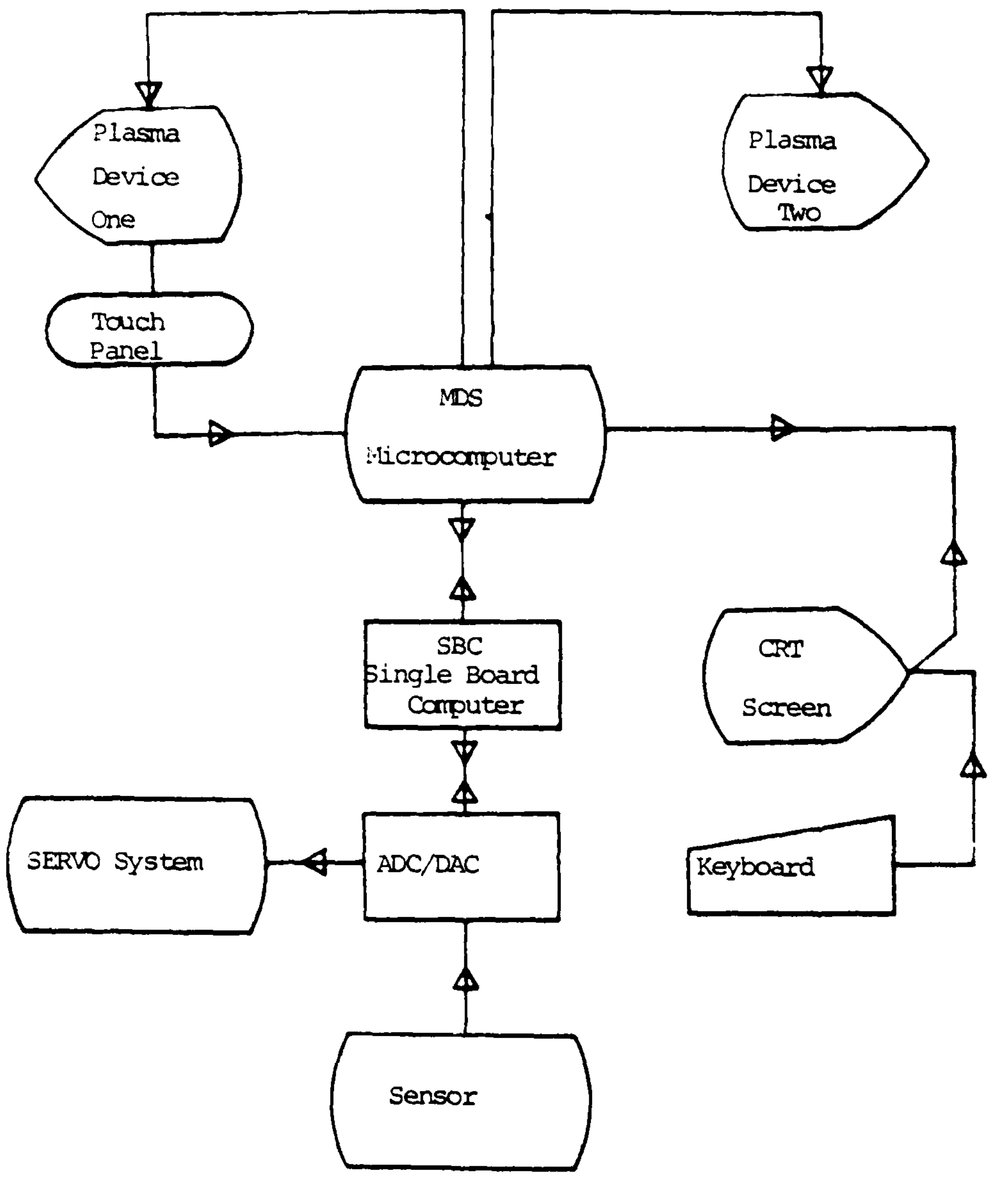


Figure 1. System Block Diagram

environment was emulated by the servomechanism and the analog plotter.

Chapter II contains the introduction to the problem and the development strategy. The Gun Fire Control (GFC) and ballistic problems and algorithms are discussed in Chapter III. Chapter IV contains hardware components.

The design of the power control system for the fire control system is given in Chapter V. Hardware and software design considerations are presented in Chapter VI. Software overview and system description are discussed in Chapters VII and VIII. The final conclusions and recommendations of this study are presented in Chapter IX.

II. INTRODUCTION TO THE PROBLEM

A. PREFACE

In order to carry out the design of a "microcomputer based gun fire control system that is controlled by an interactive display system" for shipboard tactical application; microcomputer plasma-display, ADC/DAC board and servo mechanism technologies were chosen as a technical base to design the system. The main reason for selection of these devices is that all of them are available at the Naval Postgraduate School. The shipboard weapons system was simulated by a hardware and software interface between the servo unit, sensors, ADC/DAC board, microcomputer and display system. The servo unit and potentiometers were responsible for emulating the gun and sensors system via the interface.

This study deals with the design and partial implementation of the "gun control system," the "display system," the "interface," and the establishment of a Tactical environment.

B. SCENARIO

In order for a ship to respond faster to hostile threats and to maintain operational readiness as a unit of a task force, it must be aware of the current operational environment. The operational environment is defined as the subsurface, surface and airborne contact profiles in the geographic area of interest. The contact profile consists of friendly, hostile, and unknown contacts with associated contact characteristics. Contact characteristics are data such as latitude, longitude, course, speed, range, bearing, elevation, etc.

For Naval ships not equipped with the Naval Tactical Data System (NTDS), operations performed by the Combat Information Center (CIC) during a normal peacetime watch include manual tracking of radar contacts and the solution of maneuvering board problems, In war situations, in addition to the above processes, target recognition, evaluation, designation, display and sending the target values to the weapons control central must be performed by CIC. The procedure is not only time consuming but also requires from two to four persons in peacetime and from ten to twenty persons (depending upon the type of unit) in wartime. Communication between CIC and Weapons Central is time consuming but vital for fast response time and accuracy.

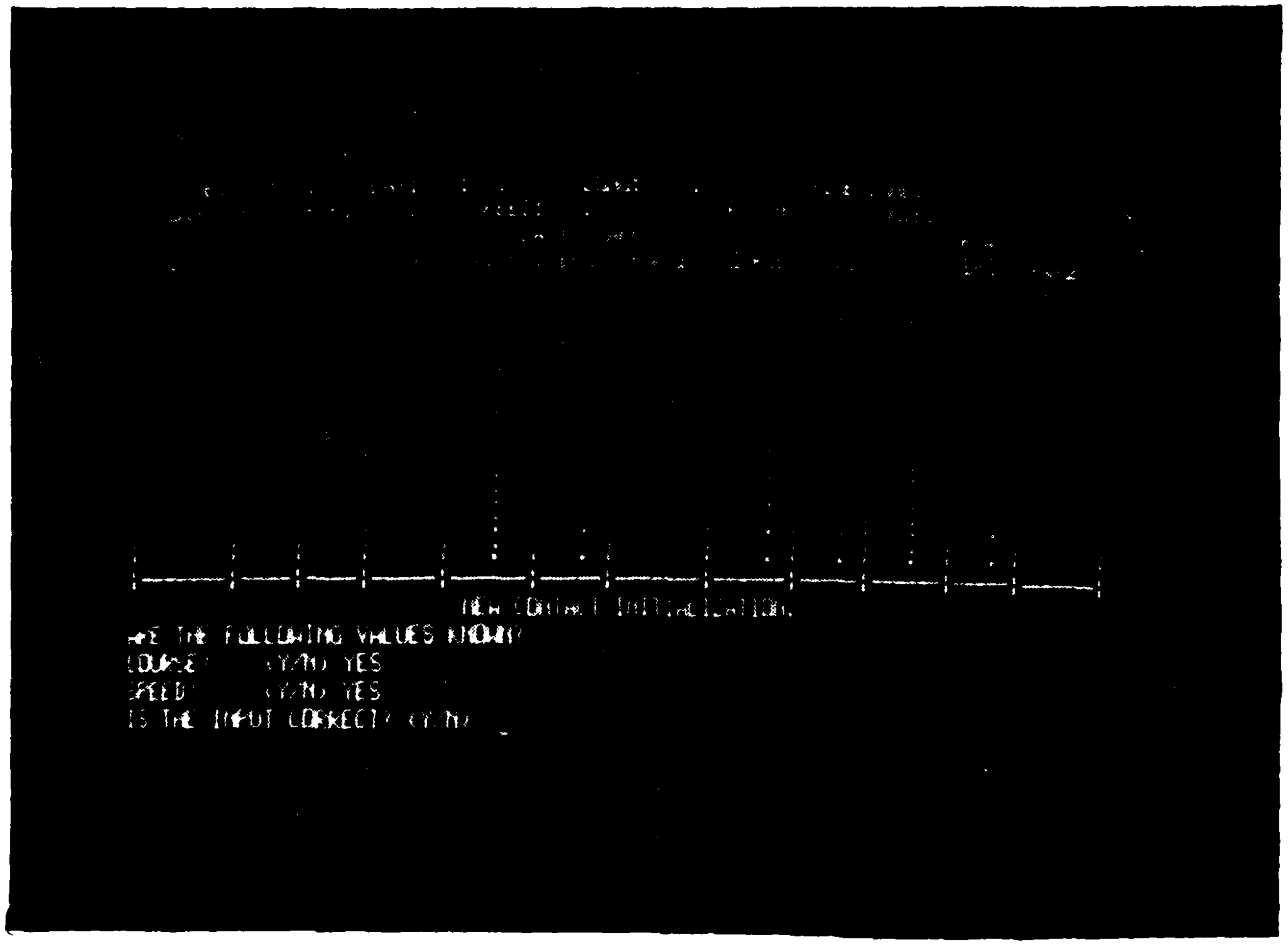
This study demonstrates the feasibility of implementing an interactive display system using microcomputer and plasma display technologies for the gun fire control system for non-NTDS type ships. The display system implemented provides the capability to present a surface, subsurface and airborne contacts profile which constitutes the operational scenario on which the display system is based. For the purpose of this study, several potentiometers were utilized to emulate the sensor (radar).

The operational scenario established for the study consisted of the sensors detecting the contacts, contact information forwarded manually or automatically to the microcomputer system which generated a surface contact profile data base for a geographic region. The data base is updated at certain time intervals that can be determined by operator and the time intervals must not be less than 17 sec. Upon the reception of the data, the display system has the responsibility for displaying the local data base.

The data base constitutes the information that is presented at the display system in alphanumeric and graphical modes. The display system has the capability to allow operator interaction to query the system for presentation of specific contact alphanumeric or graphical characteristics relative to "ownship."

The initialization of the display system begins with the setting of the system realtime clock. The operator enters: hours, minutes, seconds, time zone number, time between updates, own ship course, speed, geographic position of own ship (latitude and longitude), geographic position of grid origin (latitude and longitude) and grid scale. The CRT screen presents the operator input as shown in Figure 2.

Upon completion of the entry of the time and basic parameters of the tactical situation, the display system enters a ready state for the reception of contact data interaction. Airborne track data comes from the tracking radar (simulated by the A to D converter) and is automatically entered into the system.



```
NEW CONTACT INITIALIZATION
ARE THE FOLLOWING VALUES KNOWN?
COURSE?    (Y/N) YES
SPEED?     (Y/N) YES
IS THE INPUT CORRECT? (Y/N) _
```

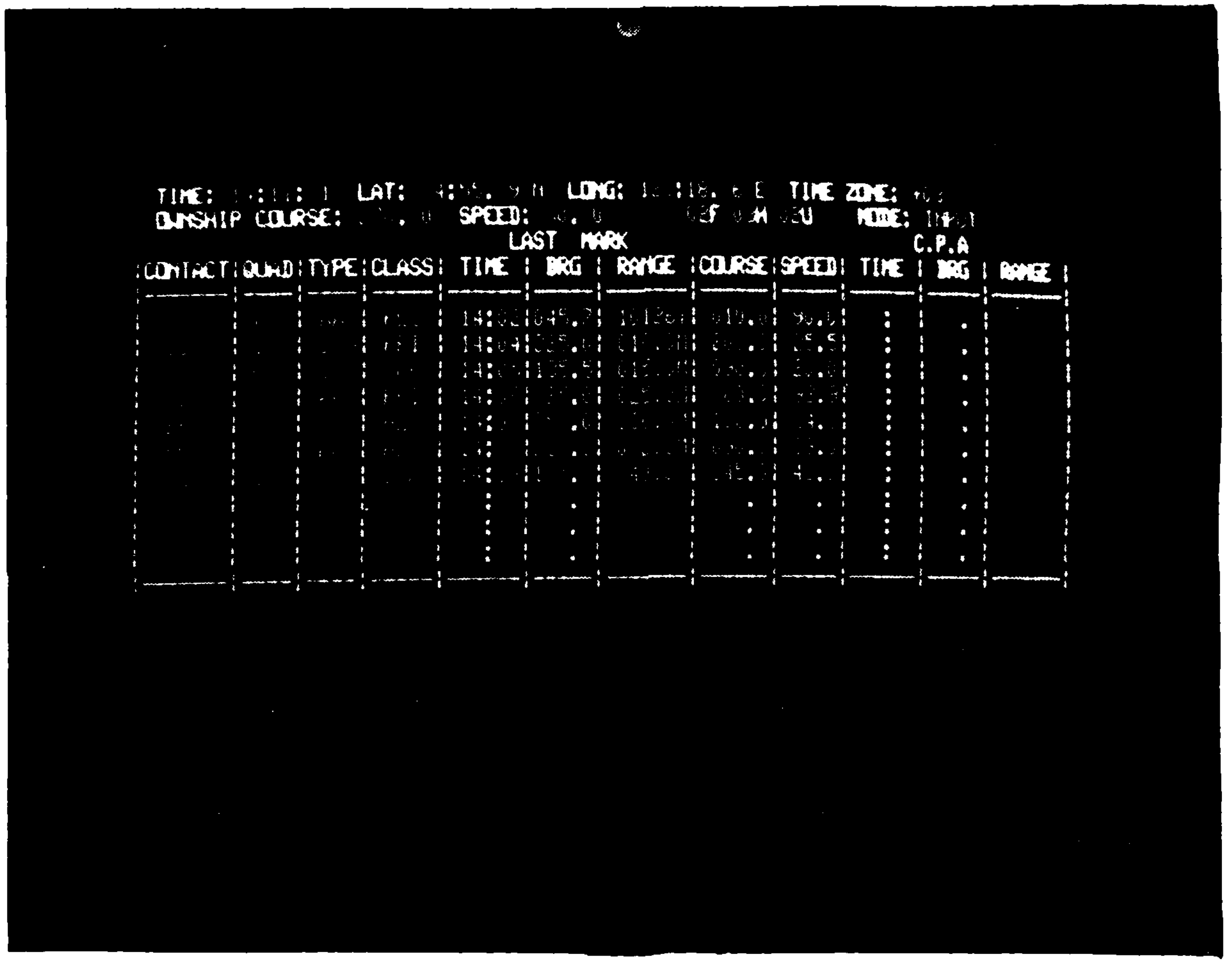
Figure 2. Initialization of Time and Display System

After the reception of inputs, the display system presents on the CRT screen the general contact characteristics as shown in Figure 3. The general contact characteristics are also presented in a graphical format at the plasma scopes as shown in Figure 4.

At any time the operator can request one of several command options from the keyboard. The command options allow the operator to interact with the display system. The operator may request general or specific contact data, set display modes, and initialize or shut down display system hardware components.

The graphical data display at the plasma scopes is presented in a primary and secondary mode. The primary plasma mode presents the surface, airborne and subsurface contact profile of interest showing the symbolic representation of contacts with vector tails representing contact speed and of course. The secondary plasma scope presents the designated target profile that is assigned to a weapon. The display shows the target in two planes and displays the trajectory and the predicted target position.

The operator has the responsibility to analyze the data presented, interact with the display system as necessary to determine both the overall contact profile and any possible threat to "ownship" and direct the gun to the target. The analysis is performed by utilizing the primary plasma display as the main source of information. The visual information



TIME: [illegible] LAT: [illegible] LONG: [illegible] TIME ZONE: [illegible]
OWNSHIP COURSE: [illegible] SPEED: [illegible] [illegible] MODE: [illegible]

				LAST MARK					C.P.A		
CONTACT	QUAD	TYPE	CLASS	TIME	BRG	RANGE	COURSE	SPEED	TIME	BRG	RANGE
[illegible]	[illegible]	[illegible]	[illegible]	14:02	045.7	[illegible]	[illegible]	[illegible]	:	.	
[illegible]	[illegible]	[illegible]	[illegible]	14:04	[illegible]	[illegible]	[illegible]	25.5	:	.	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	:	.	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	:	.	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	:	.	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	:	.	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	:	.	
				:	.		.	.	:	.	
				:	.		.	.	:	.	
				:	.		.	.	:	.	

Figure 3. General Contact Characteristics

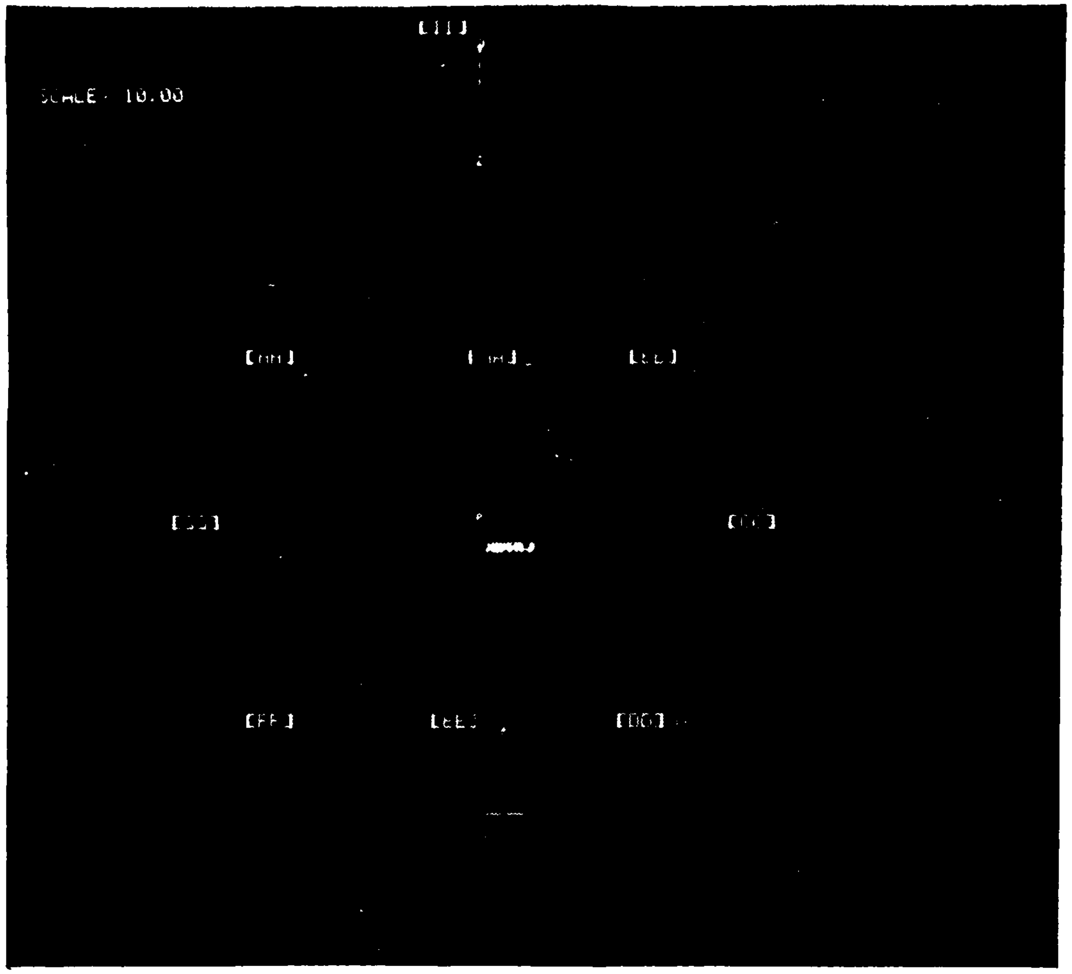


Figure 4. Graphical Representation at Plasma Scope

presented provides the operator with the necessary information to take action.

The action taken, if any, develops in the following logical sequence. The operator identifies a potential threat at the primary plasma display. The operator then uses the touch-panel feature at the secondary plasma scope to plot the threat relative to 'ownship' in surface and altitude vs. range plane if it is an air target. Simultaneously the system sends the predicted target values to the servo unit that emulates the gun which was already directed to the present target position.

At any time the operator can obtain or request contact characteristics and make changes in the initialization parameters.

C. GENERAL COMBAT INFORMATION CENTER OPERATIONS (CIC)

During the normal situation, the CIC watch team varies from two to ten or even more personnel, depending on the size of the ship as well as on the complexity of the equipment being used.

Among the problems that are normally solved by the CIC personnel, special mention needs to be made of those of plotting contacts and the determination of parameters such as course, speed, bearing, elevation and closest point of approach (CPA) of those contacts. This is a tedious and error-prone task; it often requires most of the time and effort of the CIC team. It is vitally important to detect, evaluate, display and designate the target to the weapons for combat purposes.

The processes must be done rapidly, accurately and without error. This has lead to the installation of equipment to reduce the amount of workload in the CIC, while at the same time improving the reliability of the information flow provided to the weapons central and the bridge; this equipment includes dead-reckoning devices and the NC-2 plotter.

1. Maneuvering Board Plotting Sheets

The primary responsibility of CIC is for detection, recognition, identification and designation of contacts, provision of display information for decision makers and finally for making recommendations on the tactical situation. Accordingly, CIC must supply information on all kinds of contacts within range. Contact course, speed and CPA information, is usually found using the "Maneuvering Board" plotting sheets.

The Maneuvering Board-plotting sheet (H.O. 2665-10) has been prepared in order to facilitate the solution of a ship's relative movement problem.

Although the use of the Maneuvering Board becomes straightforward after some practice, it will normally require the complete attention of one person during CIC operations.

2. Dead-Reckoning Equipment

This equipment maintains a continuous, up to the minute, geographic plot of own ships' plot in the CIC:

The Dead-Reckoning System consists of the following basic components: (1) Dead-Reckoning Analyzer (DRA); (2) Dead-Reckoning Indicator (DRI); and (3) Dead-Reckoning

Time (DRT). Course and speed inputs of own ship are fed into the DRA from Gyrocompass and pitometer-log and then to the DRT, where they cause a movable source of light to trace the ship-track continuously.

The major value of the DRT is its use in analyzing ship movements and in planning and carrying out maneuvers. As a geographic plotting device, the DRT displays true courses and allows direct computation of true contact speeds. Marking positions of the bug indicates true positions of own ship; connecting these plotted positions yields the ship's track. Plotting ranges and bearings of contacts, using own ship's positions as references, establishes their true positions. An experienced DRT operator can maintain simultaneous plots of as many as half a dozen contacts, at the same time supplying essential data (as required) on contacts that are being plotted.

A problem that is present, especially in Anti-Submarine Warfare (ASW), is that of plotting fast moving ships, and tight-turning submarines, which normally results in confused and inaccurate DRT plots. In order to help solve these problems, as well as to reduce inconvenient delays in plotting contacts because of information relayed through phone talkers, the USN Mk NC-2plotting system was developed. The NC-2 plotter consists of three major units: (1) The plotting table, (2) The Dead-Reckoning indicator, and (3) Data converter.

The main advantage of the NC-2 plotting system is its capability of receiving contact bearing and range information directly from sources (radar repeaters and the sonar). This information is then translated, and presented as colored points of light on the plotting table.

3. Anti-Air Warfare Plot and Status Boards

Airborne contacts plot on the AAW Plot boards are relative to the own ship. These boards are transparent and driven like a Maneuvering Board. Depending upon situation and the type of unit, from one to five operators plot the contact information relayed from radar operators via sound power phone. The insufficiency of the technique has become clear in case of supersonic aircraft and missile technology.

In all systems as mentioned above the target's relative position measurements are obtained from radar repeater and plotted on the systems which are read to be used. The range measurement accuracy is at best ±20 yards and bearing measurement accuracy is ±1 degrees with a well calibrated repeater and a well trained operator. The own ship speed and course are used to determine the own ship's velocity vector. The course and speed of own ship may vary ±1 degree and ±1 knot depending on the helmsman, weather conditions and the ship instruction. The combined result and the manual solution will cause an error of ±5 degrees in course and ±3 knots in speed of the target information. Also the major disadvantage of the manual solution, in addition to its poor accuracy, is the time required and the ability to obtain solutions of either the target or own ship maneuvers between radar position measurements which are typically three or more minutes apart.

4. Weapons Control Theory

In the order of occurrence, a ship must detect, display, evaluate, disseminate target information (within the ship), and if necessary destroy targets. The following is a description of the internal steps.

a. Detection. Detecting the target is the first and probably the most difficult part of the problem. Sensors are air search radars, surface radar, passive ECM (electronic countermeasure) equipment and optical devices.

b. Display. Depending on target type and class, target data are displayed in the CIC of each ship.

c. Evaluation. The process of evaluating is continuous from the moment of detection to the subsequent intercept of the target. The tools used most often are the target's relative position, its course, speed, and altitude, and its response to an IFF (identification friend or foe) challenge.

Evaluation is a process which involves a continuing appraisal or estimate of the existing air threat. It terminates with the "splashing of the target." With very little time available for detection and successful counter attack, it is mandatory that responsibilities be decentralized as much as possible. Sometimes, when confronted with a surprise attack, there is no time for the weapons officer to wait for information from CIC. The purpose of this study is to radically decrease the information relay time and response time.

d. Dissemination. Transmitting information to the other units in the group, and keeping stations on a ship informed, is the purpose of external and internal information dissemination.

All ships use sound-powered telephones to complete the communications link between CIC and weapons' battle stations.

Television equipment has done much to eliminate the excessive time consumed when transferring target data exclusively by sound powered telephones. But telephones are still used for action commands and they provide an audio backup for visual information.

e. Fire Control. Once a gun projectile is fired it follows a ballistic flight plan which is calculated by a computer but cannot be controlled by the launching ship. Effective gunfire depends on solving the fire control problem and aiming the guns correctly.

D. ALGORITHM DEVELOPMENT

Implementation of this study solves three different major problems which are: the closest point of approach (CPA), the gun fire control surface and the anti-air warfare problems.

1. The Closest Point of Approach (CPA)

K.H. Kerns and R.S. Cooper [Ref. 1] have described a way of solving Maneuvering Board problems with the aid of a microcomputer. As described in that reference, Maneuvering Board problems are divided in two basic categories.

One is the relative plot where the CPA of contacts being tracked can be calculated. The center of the plot represents the "reference" or "own" ship and any other point represents the position of a "maneuvering" ship, plotted in true bearing and range from the own ship at various times.

The other category is the vector diagram on the "triangle of courses and speeds", this allows the operator to calculate the course and speed of maneuvering ship (a contact) given the own ship course and speed, and relative course and speed of the contact (obtained from the relative plot). Solution of this problem is explained in detail in Appendix A.

2. The Surface Problem: Analytical Determination of Sight Settings

The sight-setting problem is the quantitative determination of sight angle (Vs), and sight deflection (Ds). It consists of two parts:

a. The range keeping problem.

b. Ballistic problem.

The sight setting problem is the principal part of the fire-control problem. While other factors are involved, no hits will be obtained unless the angles Vs and Ds have been correctly determined.

The best available estimate of the target distance at the present instant is present range. Its continuous determination is the range-keeping problem. Calculation of the corrections which must be included in Vs and Ds to compensate for derivations from standard conditions and for other factors, is the ballistic problem, and its solution is based on present range. Problem and solution is presented in detail in Chapter III.

3. The Anti-Air Problem: Analysis of Sight and Fuze Settings

Since an airplane may be climbing or diving, target motion is not necessarily horizontal. Present target position with respect to own ship is established by the three coordinates: present range (R), relative target bearing (Br), and target elevation (E). R, Br and E are provided by radar. In this study they are simulated by different potentiometers.

The direction of motion in a horizontal plane is described by target angle (A), which is measured in a horizontal plane. Target speed is defined by vertical speed or rate of climb (dH), and air target horizontal ground speed (Sh) that usually is called target horizontal speed. These two speeds are simply components whose resultant is actual target speed relative to earth. The use of Sh and dH simplifies the positioning-problem solution, for they indirectly define the inclination of target motion to the horizontal. It should be noted that dH is vertical speed, being equivalent to the rate of change of height (H). It also represents the total vertical relative position since own ship moves only on the surface.

It is not practicable to obtain basic coordinates by direct measurement at each instant, for any interruptions or errors in individual measurements would be reflected in the computed values of sight angle (Vs), sight deflection (Ds), and fuze setting (F). Generally computers generate smoothed target position gathered by several measurements and this position is used as the basis of the solution for Vs, Ds and F. The problem and detailed solution are presented in Chapter III.

F. DEVELOPMENT PLAN

Upon completion of a 'top-down analysis' of the effort required to implement the system, the project was divided into three major areas:

1. Hardware and software interfaces required to control the position and the speed of the Modular Servo System.
2. Hardware and software functions and capabilities to be developed for the MDS Microcomputer.
3. Hardware and software interfaces required to provide sensor to computer (from the potentiometer, ADC/DAC board, SBC board to computer) communications between the ADC/DAC board and the MDS Microcomputer.

From the analysis, a critical development path was determined creating a development order (priority, time basis) for the three major areas of the project.

One of the critical project areas is the software and hardware interfaces between the MDS microcomputer system to Single Board Compter (SBC). This is provided by using "programmable peripheral interface" for parallel port communication between the MDS and SBC. The interface is explained in detail in Chapter VI.

At completion of the interface, the effort was directed to the other two proejct areas in a Flip-Flop manner allowing interim testing of the partial system configuration. Greater emphasis was placed on the hardware and software development at the MDS end, because of the greater proportion of development effort required in this project area. The iterative flip-flop development was highly desirable to provide feedback to the development process. The iterative process allowed the development of an adequate test data set to test and validate the system design upon completion.

The project development was evaluated under the following guidelines:

- All hardware components and interfaces were tested on a stand-alone basis before incorporating them into the system design being implemented.
- Software development was approached in such a manner as to allow testing at the procedure and module level.
- Upon completion of the system design implementation, the system was evaluated utilizing a test data set to demonstrate system capabilities compared against design objectives.

III. GUN FIRE CONTROL, BALLISTIC PROBLEMS AND ALGORITHMS

A. THE SURFACE PROBLEM: ANALYTICAL DETERMINATION OF SIGHT SETTINGS

The sight-setting problem is the quantitative determination of sight angle (Vs), and sight deflection (Ds). It consists of two parts: (1) The range keeping problem and (2) Ballistic Problem. The sight-setting problem is the principal part of the fire-control problem, for while other factors are involved, no hits will be obtained unless the angles Vs and Ds have been correctly determined. Graphic representation of factors relating to the sight-setting problem is shown in Figure 5.

1. The Range Keeping Problem

The best available estimate of the target distance at the present instant is present range, and its continuous determination is the range-keeping problem. The purpose of this chapter is the theoretical determination of sight settings for the surface fire-control problem.

a. Relative Target Motion

In almost all cases the distance to the target is continually changing. There are two reasons why this distance, or range, must be kept. First, the instant of measurement comes before the instant of firing, and during the interval, the range varies. Second, range measurement may be interrupted. In either event, present range is required for the correct determination of Vs and Ds.

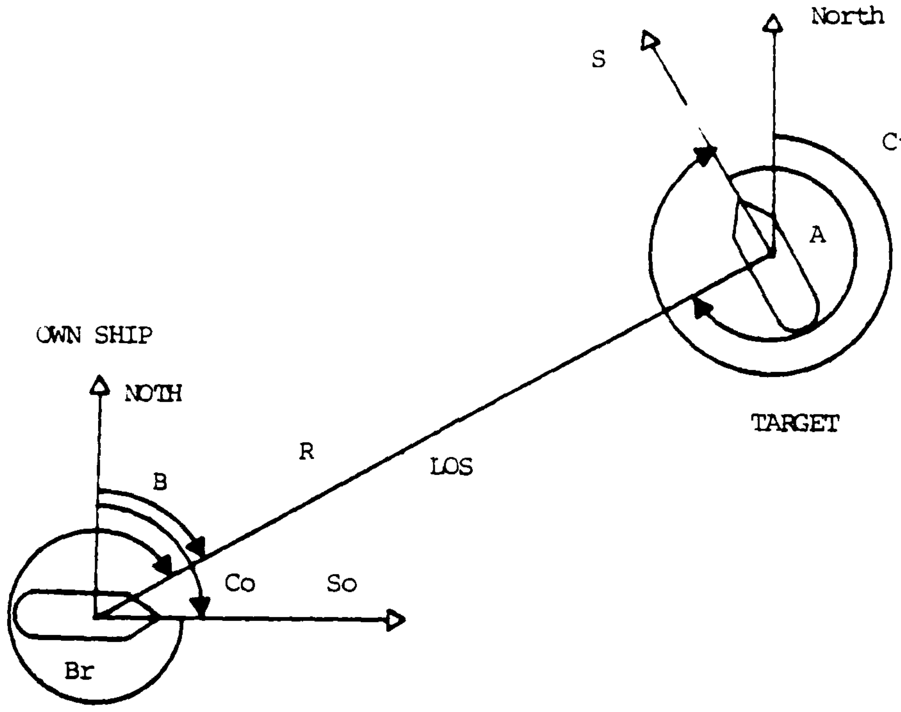


Figure 5. Graphic Representation of Factors Relating to the Sight-Setting Problem

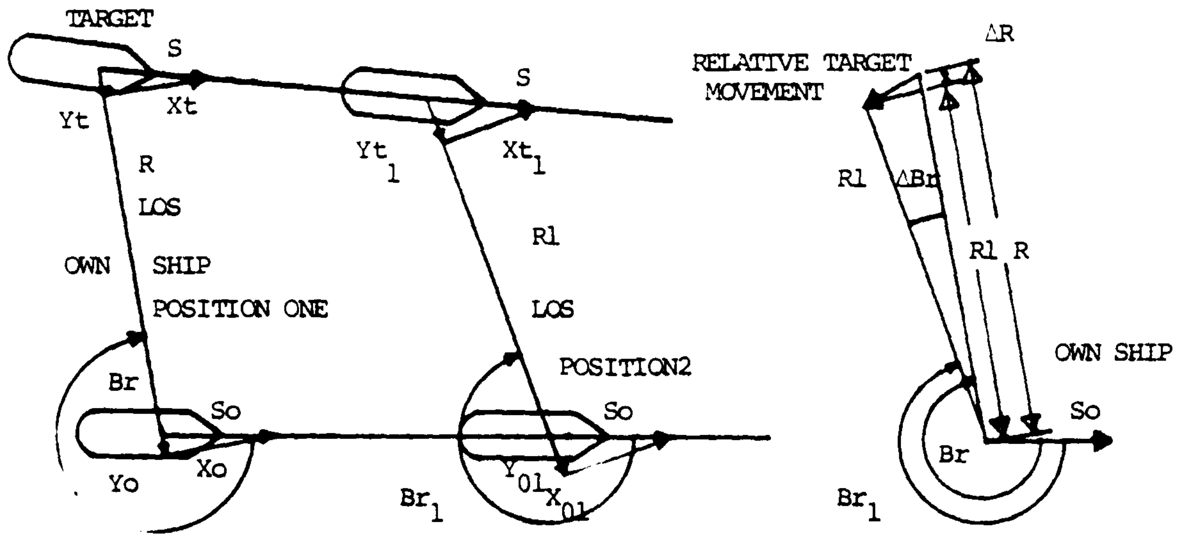


Figure 6. Relative Target Movement

Relative target motion is the apparent motion of the target with respect to own ship, and is due to motions of both own ship and target. The target position with respect to own ship is established by the distance to the target along the line of sight, or present range (R), and by the target bearing from own ship, or relative target bearing (Br).

The problem of range keeping is to continuously position the target by keeping track of the values of R and Br, which are always changing except in very special and rare circumstances. While R is the quantity of principal importance in the determination of sight settings, the continuous determinations of R and Br are so intimately related that they must both be examined together. Relative target motion is shown in Figure 6.

b. Evaluating Relative Target Motion

The analysis of the problem is most conveniently made with the aid of a vector diagram in which the motions of own ship and the target are represented by vectors plotted with respect to the line of sight. The range-keeping diagram is shown in Figure 7.

The usual procedure in determining the components in and across the LOS is using vectors and geometry. The values of components are:

$$Yo = So \cos Br$$

$$Yt = S \cos A$$

$$Xo = So \sin Br$$

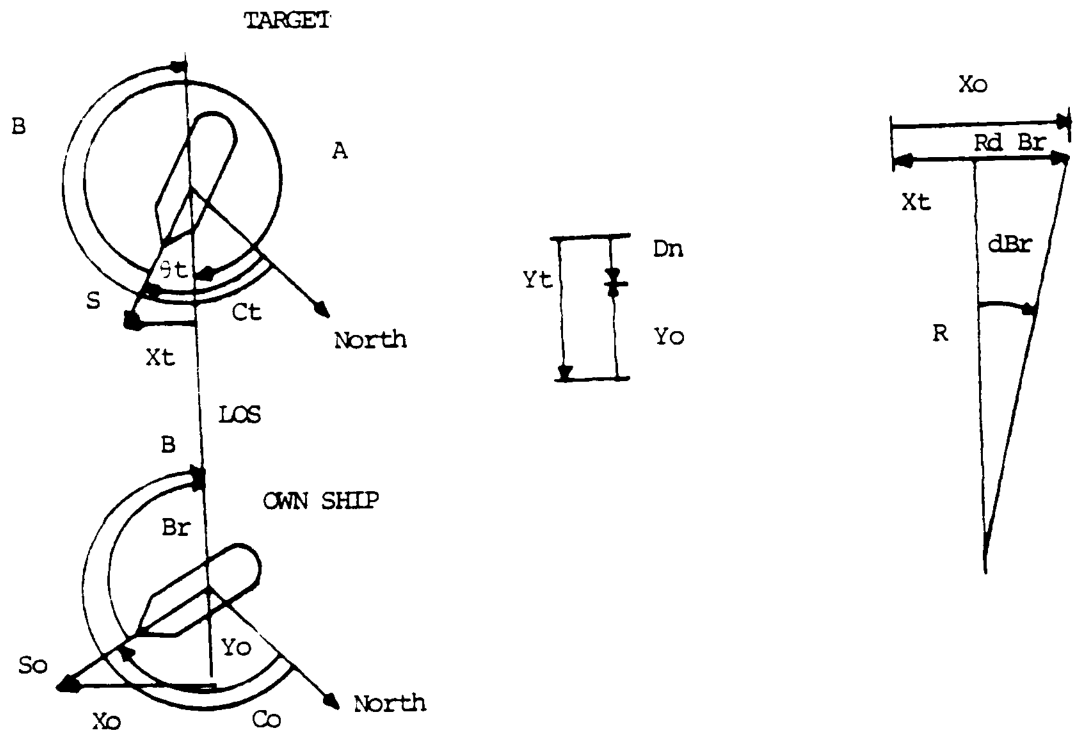


Figure 7. Range-Keeping Diagrams

$$Xt = S \sin A$$

where

Yo	=	Line component at own-ship velocity
Yt	=	Line component at target velocity
Xo	=	Gross component of own-ship velocity
Xt	=	Gross component at target velocity
A	=	Target angle
Br	=	Relative target bearing
So	=	Own-ship speed
S	=	Target speed

The quantities are shown in Figure 7.

Range rate (dR) is the algebraic sum of Yo and Yt. A line component is called decreasing when it tends to decrease the range, increasing when it tends to increase the range shown in Figure 7b.

$$dR = Yo + Yt$$

The linear bearing rate RdBr is the algebraic sum of Xo and Xt, and must be established before the angular rate can be determined. Each X component is decreasing when it causes a decrease in Br and increasing when it causes an increasing Br. If the algebraic sum of Xo and Xt is decreasing, RdBr is decreasing and Br is decreasing. If the sum is increasing, Br is increasing as shown in Figure 7.

$$RdBr = Xo + Xt$$

The angular rate should be expressed in minutes of arc per second. The transition from linear to angular measure is accomplished by dividing the linear rate by the range rate, using the proper units.

$$dBr \text{ (minutes of arc per second)} = \frac{1936 \times RdBr}{R}$$

where

dBr = Relative angular-bearing rate.

Range can be kept by multiplying the rates by elapsed time to obtain increments of range (ΔR) and bearing (ΔBr). When these increments are added algebraically to the initial values of R and Br, new values are obtained. New rates, and corresponding new increments can be computed on the new R and Br to keep the range and bearing during the next interval of time. By making the process continuous, present range can be generated with no further measurements after the initial determination. Thus, present range will be continuously available even though there may be interruptions in ranging.

2. The Gun Ballistic

a. The Ballistic Problem

The determination of sight angle (Vs) and sight deflection (Ds), based on present range, is the ballistic problem. This involves the evaluation of the elements which, when combined, give the values of Vs and Ds that will result in a hit.

There is an angle of departure for standard conditions corresponding to each possible target distance, or present range, within the limits of maximum range of a gun. Standard conditions are never obtained, and compensating corrections must be applied to the standard angle of departure to obtain the value of Vs which will provide the trajectory required to hit a target at a given range. Similarly, corrections must be computed to obtain the correct value of Ds.

b. The Gun Ballistic Problem

The gun ballistic problem takes into account the calculable corrections which must be included in Vs and Ds, under the normal conditions of aiming a gun with respect to a line of sight which coincides with the line of position.

Range tables provide the data required for the computation of the elevation angle depending upon the type of gun. The range to use in entering the range table is the best available estimate of the target distance at the instant of rounds impact.

c. Errors

Gun ballistic characteristics test of the corrections to compensate for the following variations from standard conditions:

(1) Group A errors. The elements causing errors notdependent on target bearing are

(a) Variation in initial velocity (I.V) due to powder temperatures.

(b) Variations in I.V due to erosion.

(c) Variations in air density.

(d) Drift.

The first three of these elements cause errors in range, the fourth is an error in deflection.

(2) Group B Errors. Group B errors include elements causing errors which are dependent on target bearing.

(a) Wind.

(b) Motion of target.

(c) Motion of own ship.

Each of these factors causes an error in range and deflection, for each generally has components in and across the LOS.

All these calculations and corrections were done in SWF (Surface warfare) program module.

B. ANTI-AIR PROBLEM: ANALYSIS OF SIGHT AND FUZE SETTINGS

The calculation of sight angle and sight deflection in the AA problem are similar in many respects to the corresponding calculations in the surface problem. The AA has the same distinct parts, namely: (1) continuous target positioning, and (2) continuous ballistics computing. The target's freedom of movement in three dimensions and the use of time fuzed projectiles introduces additional complications. Since sight and fuze settings are based on the instantaneous relative target position, the positioning part of the AA problem is presented first.

1. <u>Positioning Computations</u>

a. Relative Target Position

Present target position with respect to own ship is established by the three coordinates: present range (R), relative target bearing (Br), and target elevation (E). These quantities are shown in Figure 8. R, E and Br are provided by radar. In this study they are provided by different potentiometers whose input of ADC/DAC board (simulation of sensor).

It is not practicable to obtain the basic coordinates by direct measurement at each instant, for any interruptions or errors in individual measurements would be reflected in the computed values of sight angle (Vs), sight deflection (Ds), and fuze setting (F). As usual in this project, the computer generated relative target positions, and these positions are used as the basis of the solution for Vs, Ds and F.

b. Target-Motion Designation

Since an airplane may be climbing or diving, target motion is not necessarily horizontal, and three quantities are needed to define it, as shown in Figure 8a.

The direction of motion in a horizontal plane is described by target angle (A), which is measured in a horizontal plane just as in the surface problem The target speed is defined by vertical speed or rate of climb (dH), and air target horizontal ground speed (Sh) usually called target horizontal speed. These two speeds are simply components

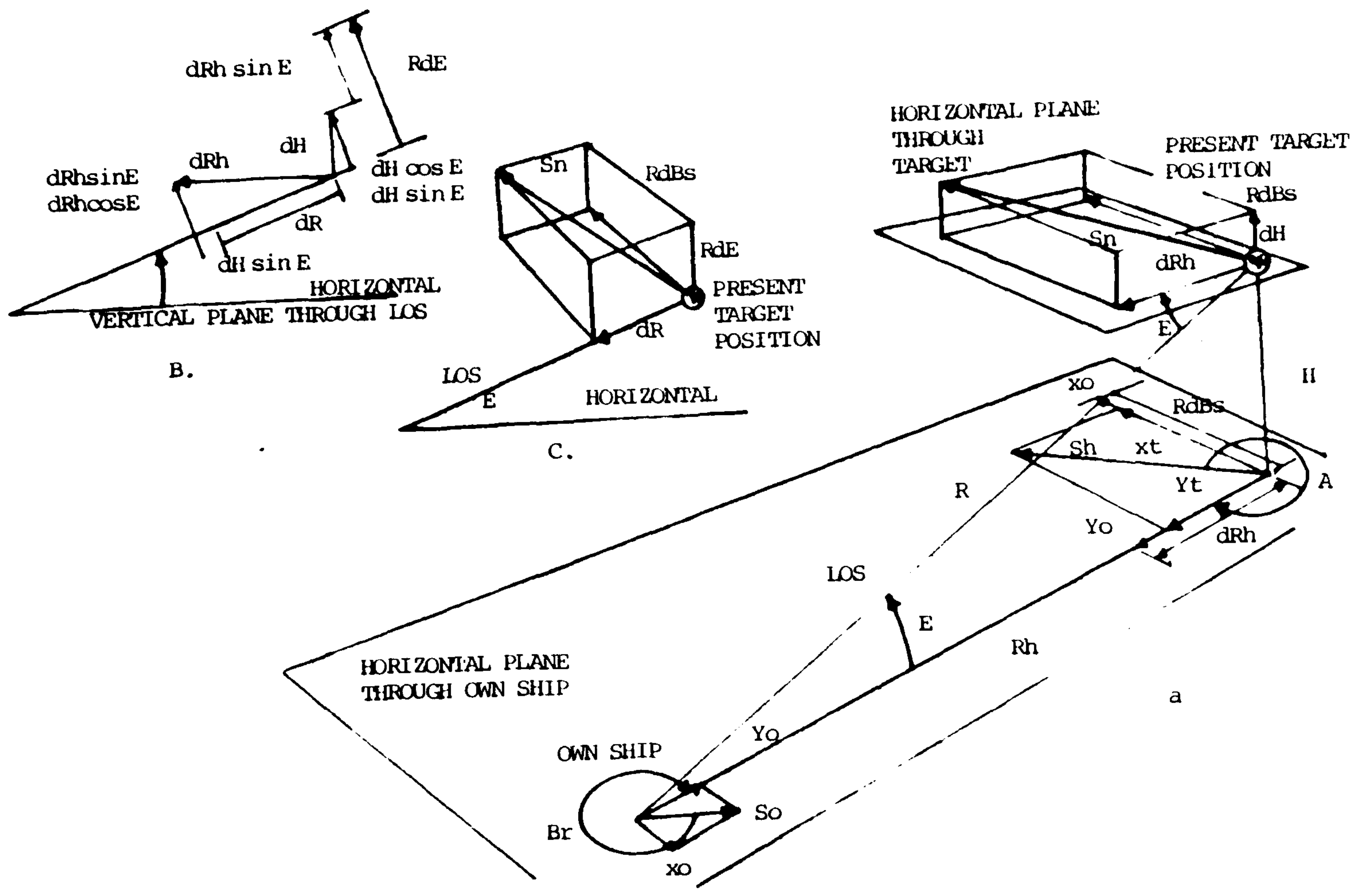


Figure 8. Relative Target Motion Resolution

whose resultant is actual target speed relative to earth. The use of Sh and dH simplifies the positioning-problem solution, for they indirectly define the inclination of target motion to the horizontal. It should be noted that dH is a vertical speed, being equivalent to the rate of change of height (H). It also represents the total vertical relative motion, since own ship moves only in the horizontal.

In the project, target speed components are simulated by potentiometers and scaled by S/W programs.

c. Relative Target Motion

For Anti-Air Fire Control relative target motion is used to keep track of relative target position in a manner similar to that employed for Surface Fire Control. In other words, the present values of R, Br, and E are generated. This requires the determination of range rate (dR), relative angular bearing rate (dBr), and angular elevation rate (dE). These rates are multiplied by time increments (ΔT) to obtain generated-range increments (ΔcR), generated-bearing increments (ΔcBr), and generated-elevation increments (ΔcE). The generated increments are added to the initial values of their corresponding coordinates (jR, jBr, and jE) to obtain the generated values cR, cBr, and cE. Thus:

$$cR = jR + \Delta t \times dR = jR + \Delta cR$$

$$cBr = jBr + \Delta t \times dBr = jBr + \Delta cBr$$

$$cE = jE + \Delta t \times dE = jE + \Delta cE$$

The initial values are obtained by sensors (simulated by potentiometers). The generated increments are computed by the MDS system to which the change rates must be added. The three rates dR, dBr, and dE completely describe relative target motion and are determined by following these two steps:

1. Resolve own-ship and target motions into horizontal and vertical components of relative target motion.

2. Resolve the horizontal and vertical components into the three rates with respect to the line of sight (LOS).

d. Horizontal and Vertical Rates

Since own-ship motion and part of the target motion are defined by horizontal speeds and directions, it is a simple process to determine relative horizontal target motion.

The line components (Yo and Yt) and cross components (Xo and Xt) can be computed by using Br and A. Their effects upon range and bearing may be considered as increasing or decreasing as was done in the surface problem. The components are combined to produce horizontal range rate (dRh) and linear deflection rate (RdBs) as shown in Figure 8a.

Mathematically, the components may be computed from these equations:

$$Yo = So \cos Br$$

$$Yt = Sh \cos A$$

$$Xo = So \sin Br$$

$$Xt = Sh \sin A$$

Horizontal range rate is conventionally considered to be negative when it decreases the range, consequently the expression for horizontal range rate is

$$dRh = -(Yo + Yt)$$

Linear deflection rate (RdBs) is algebraic sum of Xo and Xt.

$$RdBs = Xo + Xt$$

The vertical relative motion is the rate of climb (dH) as previously mentioned and is positive when the target is climbing, negative when diving. Hence total relative target motion is defined by dRh, RdBs and dH. When these three rates are combined about the target position, their resultant is the vector Sr, whose direction represents the path and whose length represents the speed of the target with respect to own ship.

e. Rates In and About the LOS

The horizontal and vertical rates are simply employed in determining the rates at which the basic coordinates R, Br and E are changing. It is helpful to recognize the existence of the vector Sr, although it is not used directly in maintaining target position. This single vector can be resolved into any number of components but those needed in generating target position are:

(1) Range rate, measured in or along the LOS,

(2) Bearing rate, measured perpendicular to the LOS in a horizontal plane,

(3) Elevation rate, measured perpendicular to the LOS in a vertical plane. The relations of the linear rates to Sr are shown in Figure 8c.

Range rate along the LOS is direct range (dR). Since the LOS is elevated dR is not the same as dRh, but has a component due to dRh and dH, as shown in Figure 8b. Both dRh and dH lie in the vertical plane through the LOS and are inclined to the LOS by E and (90 - E), respectively. It can be seen from the Figure 8c that:

$$dR = dH \sin E + dRh \cos E$$

The linear elevation rate (RdE) also has components due to dRh and dH. It may be expressed as:

$$RdE = dH \cos E - dRh \sin E$$

The angular elevation rate (dE) is obtained by dividing the linear rate by present ragne (R). Thus:

$$dE = \frac{R\,dE}{R}$$

The linear relative bearing rate (RdBs) is measured perpendicular to the vertical plane through the LOS, and has the same value whether measured in the the slant plane or in the horizontal. Since relative target bearing (Br) is measured in the horizontal, relative angular bearing rate (dBr) must be computed for the horizontal by dividing RdBs by horizontal range (Rh). It is evident that Rh = R cos E, hence:

$$dBr = \frac{RdBs}{R \cos E}$$

2. Ballistic Computations

The purpose of generating present target position is to provide a formulation upon which to base the calculations of sight angle, sight deflection, and fuze setting. These are developed from present angle (R), relative target bearing (Br), and target elevation (E). Also we should consider that an effective solution must be continuous and instantaneous; such a solution is provided by the MDS system.

a. Statement of the Problem

A gun fired under a given set of conditions will produce just one trajectory that is in no way affected by target motion or by the point on the trajectory at which the hit or burst occurs. The target may be at any point in a given trajectory. Therefore, the problem is different than the surface problem (calculation of time of flight).

No matter where the target is on the trajectory, the angle of the departure or vertical gun elevation (Eg) is the same. However, sight angle (Vs) and sight deflection (Ds) will have different values for each point on the trajectory. Evidently Vs and Ds dependent upon both present range (R) and target elevation (E). Since corrections for wind, and own-ship and target motions depend upon relative target bearing (Br), sight settings in any practical case also depend upon Br. The problem is to determine the values of Vs and Ds which will place the bore axis in the space position that will produce the trajectory required to hit the target.

While in theory the correct values of Vs and Ds will produce hits, a perfect evaluation of these two angles is impossible. Some errors inherent in the solution methods and others due to inaccurate estimates of such factors as air density, wind and target motion are bound to exist. In addition, natural gun fire dispersion causes variations between shots fired under seemingly identical conditions. Consequently, projectiles are fuzed to burst near the plane to increase the hitting probability. When the fuze bursts the projectile close to the target, some of the fragments may hit the plane regardless of small discrepancies in sight settings. In addition, the nearby blast effect may either damage the plane or at least make it difficult to maintain a smooth, uninterrupted bombing run. So the determination of fuze setting (F) is also a part of the problem when time fuzed projectiles are used.

b. Ballistic Factors

In anti-air fire ballistic problem has the following factors:

- Gravity
- Drift
- Target motion

- Own-ship motion
- Wind
- I.V. variations
- Air-density variations.

Effects of these factors depend on the type of gun. Gun ballistic data that is generally restricted information.

c. Relative Target Motion

During the time of flight (Tf) the target moves from the position it occupied at the instant of firing. It is obvious that the gun should be fired not at the present position but at the advance position the target will occupy at the end of Tf. However, the LOS must be directed at the present position hence the gun must lead the LOS (and the target) by correction or elevation and deflection.

d. Relative Target Movement

The time of flight movement of the target relative to own ship is Tf $\times$ Sr as shown in Figure 9. The velocity Sr is simply the resultant of actual own-ship velocity and actual target velocity. As long as own ship and target hold their courses and speeds, the vector Sr is fixed in space. In other words, its value is not instantaneous like the values of the rates dR, RdBs, and RdE. These three rates have instantaneous values only because they are measured with respect to the LOS, which is continuously changing position with respect to own-ship and target.

The vector Sr can be resolved into many different sets of components that will completely describe it, and the

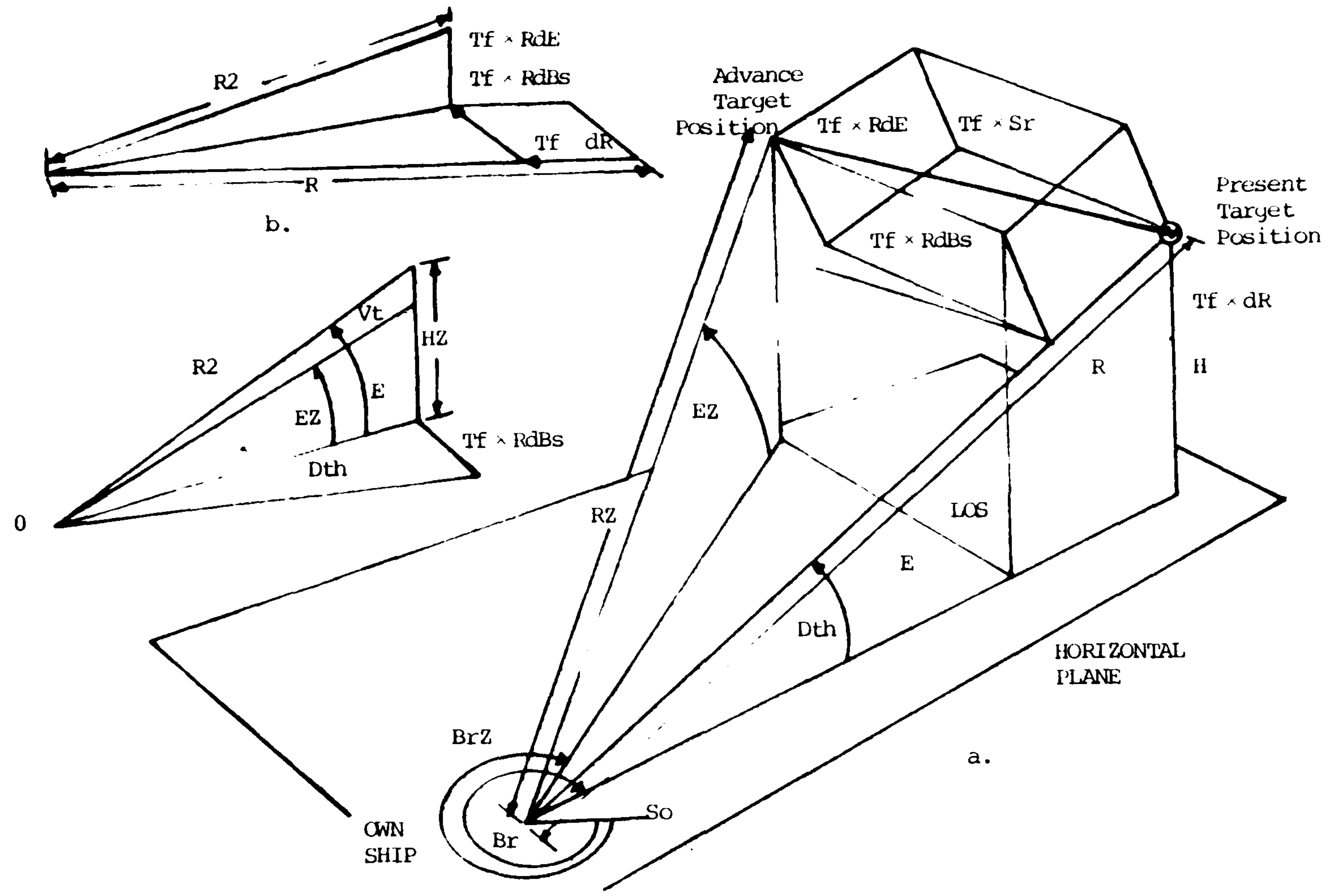


Figure 9. Advance Target Position and Predictions

components of any chosen set will remain fixed in value so long as Sr remains constant. The values of the rates dR, RdBs, and RdE for the present target position form one set of these components, and are already available from the positioning part of the problem. When the present values of rates are each multiplied by Tf, their products are components of relative target movement during Tf. When these components are added vectorially as in Figure 9, they terminate at the end of the vector Tf × Sr and define the advance target position with respect to the present target position.

e. Advance Target Position

The target position relative to own ship that will exist at the end of the time of flight is the advance (or predicted target position). Like the target position, it is defined by these coordinates:

(1) Advance (or predicted) range (R2)

(2) Predicted relative target bearing (Br2)

(3) Predicted target elevation (E2).

These quantities are predictions of the "present" values of R, Br, and E that will eixst at the moment a projectile, fired at the present instant hits or bursts; i.e., at the end of the time of flight. The numeral 2 is used to indicate predicted values pertaining to the advance position.

The advance position is obtained by the application of various plane geometry and plane trigonometry relations.

The correct value of R2 can be computed by using the plane geometry theorem direct equation.

$$R2 = ((Tf \times RdE)^2 + (Tf \times RdBs)^2 + (R + Tf \times dR)^2)^{1/2}$$

The change in height during time of flight is evidently Tf dH, since dH is the total vertical-motion component.

The change in height during time of flight is evidently Tf dH, since dH is the total vertical motion component. Thus the height of advance target position, or H2, in Figure 9c, is

$$H + Tf \times dH .$$

Since $H = R \sin E$

$$H2 = R \sin E + Tf \times dH$$

Predicted target elevation (E2) is evidently the angle subtended at the gun by the predicted height (H2). Thus:

$$\sin E2 = \frac{R \sin E + Tf \times dH}{R2}$$

The elevation prediction (Vt) can be readily obtained from the relation

$$Vt = E2 - E$$

The vertical gun elevation (Eg) must of course include the prediction Vt to allow for the vertical change in target position during the time of flight. The value of Vt is:

$$\sin Vt = \frac{Tf \times RdE}{R2}$$

or

$$Vt = \frac{\sin^{-1} Tf \times RdE}{R2}$$

and finally predicted elevation is:

$$E2 = E + Vt = E + \frac{\sin^{-1} Tf \times RdE}{R2}$$

The change in relative target bearing during Tf is the horizontal angle Dth, subtended by the horizontal distance between the advance target position and the vertical plane through the LOS. It is clear from Figure 9c that

$$\sin Dth = \frac{Tf \times RdBs}{R2 \cos E2},$$

and

$$Dth = \sin^{-1} \frac{Tf \times RdBs}{R2 \cos E2}$$

The sign of Dth is the same as that of RdBs, which has a negative value in the illustration, since it tends to decrease. Hence predicted relative bearing is:

$$Br2 = Br + Dth$$

f. Time of Flight

It has been stated that range, wind, I.V. variations, and air-density variations affect time of flight, hence Tf should include corrections for each of these factors.

Standard time of flight may be obtained from the trajectory graph or calculated by a special algorithm when entered with R2 and E2. In this study the following algorithm is used. Initial velocity horizontal component is

$$I.V.x = I.V. \times \text{Cos } E2$$

Horizontal component of range is

$$R2x = R2 \times \cos E2$$

thus time of flight is

$$Tf = \frac{R2x}{I.Vx}$$

g. Fuze Settings

It should be obvious that the setting of a time fuze must be based upon the time its projectile will be in flight. In this study mechanical fuzes are assumed so that air-density variations do not effect mechanical fuzes and hence their settings are the same as the time of flight.

IV. HARDWARE COMPONENTS

Given below is a list of the Hardware components used in this study and discussed in this chapter.

1 MDS Microcomputer
1 SBC Board
1 ADC/DAC Board
1 Modular Servo System
2 2500 Plasma-Scope Gas Discharge Display System
1 Plasma-Scope Touch Panel
1 Datamedia Elite Video Terminal
6 Potentiometers

A. MICROCOMPUTER DEVELOPMENT SYSTEM "MDS" DESCRIPTION

The Intellec microcomputer development system (MDS) is designed around Intel's popular 8080 microprocessor. The MDS utilizes the INTEL SYSTEMS IMPLEMENTATION SUPERVISOR (ISIS-II), as its operating system in conjunction with the INTELLEC system "Firmware Monitor" package.

The 8080 has a 2-usec instruction cycle, a repertoire of 72 powerful instructions, unlimited subroutine nesting, and a versatile interrupt scheme. The 8080 supports up to 65,536 (64K) words of memory and up to 512 I/O devices (256 input, 256 output). The basic hardware configuration includes 65,536 (64K) bytes of Random-Access-Memory (RAM), and six fully implemented I/O interfaces to:

- a Teletype (including its paper tape reader),
- a CRT terminal (or other compatible device),
- a high-speed paper tape reader,

- a high-speed paper tape punch,
- a line printer, and
- Intel's Universal PROM Programmer.

The "standard" configuration of the overall computer system consists of the MDS microcomputer, a dual-diskette drive (a quarter million bytes per floppy disk), a CRT and/or teletype for man-machine interface, a resident high-level PL/M-80 compiler, and resident assembly language 8080/8085 macro assembler.

B. PLASMA PANEL DESCRIPTION

This section describes the SAI Technology Company's Model 2500 Plasma-scope Gas Discharge Display System, which is capable of displaying alphanumeric characters and/or graphics. The plasmascope also features several configuration options to provide full capability for interfacing with a keyboard and various parallel and serial computer interfaces such as:

- The Interface I/O.
- The Display Buffer.
- The Vector Generator.
- The Character Generator.
- The Manual-Entry Keyboard.

The plasma panel contains 262,144 individual dots which are capable of being discretely addressable in terms of selecting specific x and y coordinate values for excitation; e.g., to create or extinguish light. The panel is normally driven by selecting parallel groups of lines on one axis (Y) and scanning on the other (X). This operation provides displays of alphanumeric data using a data matrix/format for characters and symbols. Also, by selecting single element location in coordinated fashion, graphics can be created on the display surface.

More specifically, the plasma panel consists of two panels of clear glass each of which has embedded parallel electrodes that are appropriately separated. The panels, aligned with the electrodes at 90 degrees, are separated by a dielectric and space seal. This spacer area is filled with a neon-based gas.

The model 2500 Plasmascope has the following interface capabilities:

- Parallel I/O Buffer, 16 bits.
- Differential I/O Buffer.
- Serial I/O Buffer.
- Data Channel Controls.
- Synchronous Interface.
- Asynchronous Interface.
- NTDS-3V I/O.

C. PLASMASCOPE TOUCH PANEL DESCRIPTION

The touch panel is an input device for Plasmascope which allows the operator to touch the display panel and convey positional information to the computer. The touch panel uses a crossed array of light beams projected just above the display surface. When an x and y beam is broken by an obstacle such as finger, the panel reads the x-y address into the computer. The panel then waits until the finger is repositioned before sending a new address.

There are 16 horizontal and vertical light beams that create a grid pattern of 256 positions which can be identified by the touch panel logic; this logic transforms this positional data to an eight bit data word.

The word consists of 4 bits representing the horizontal position and 4 bits representing the vertical position (x and y coordinates respectively, referenced to the left upper corner). These two 4-bit nibbles in conjunction with the touch-panel status bit, makes the positional information available to the user for any compatible I/O device.

D. ANALOG INPUT/OUTPUT SYSTEM

These microcomputer peripherals provide two functions that interface directly to Intel's SBC80 and Intellec MDS microcomputers. The functions are: (1) Analog Data Acquisition and (2) Analog Output. The devices are electrically and mechanically compatible with any SBC80 and Intellec MDS. Both analog input and output systems are contained on a single printed circuit board that is treated as memory input or output by the CPU. The cards will mate to any memory or I/O slot. The analog interface for each system is a connector at the opposite edge of the board from the bus connector.

The Data Acquisition system is available with up to 64 channels single-ended on one board. It includes an input multiplexer, high gain instrumentation amplifier, 8-bit A/D converter along with all the necessary timing, decoding and control logic. A DC/DC converter (+5 to ±15V) is also available so that only the computer's power supply is required. The Data Acquisition System is available with two optional 8-bit D/A converters to provide analog input and output on the same board.

E. SERVOMECHANISM DESCRIPTION

The Feedback MS150 Modular Servo system is particularly designed for experimental use by students and technicians who are studying closed loop control systems, and who are concerned with obtaining a general qualitative group of closed-loop techniques which complement analytic investigations. Components of the Closed-loop system servomechanism are shown in Fig. 10. The components of the MS 150 DC System are:

- Operational Unit
- Attenuator Unit
- Pre-Amplifier Unit
- Servo Amplifier
- Power Supply
- Motor Unit
- Input Potentiometers
- Output Potentiometer

Each of the units of this equipment is fitted with magnetic feet and can be attached to the base board in any desired position.

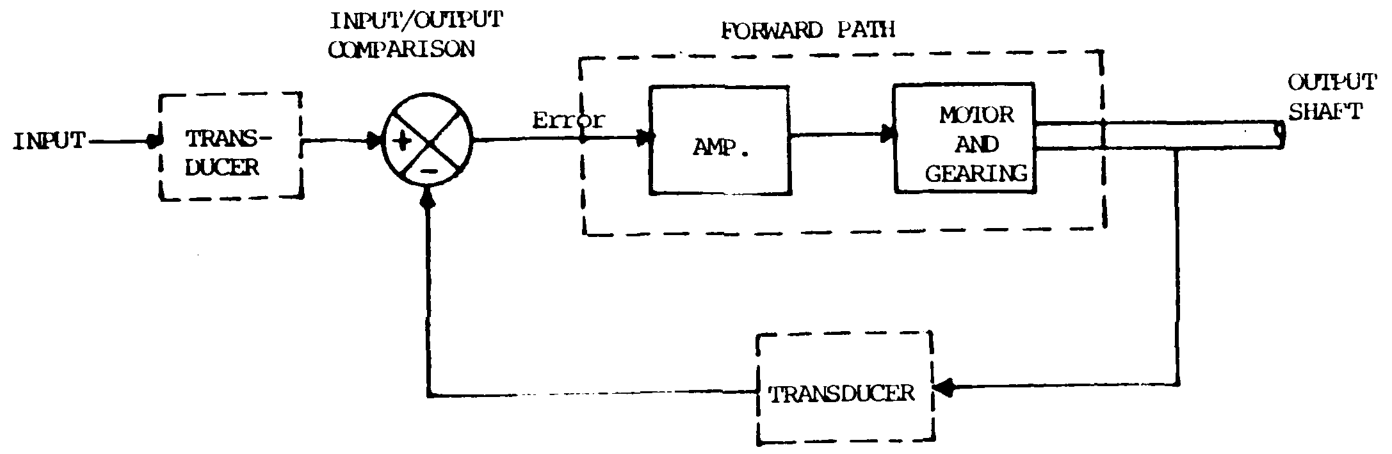


Figure 10. Elements of a Closed-Loop System Servomechanism

The main power supplies for the Servo Amplifier Unit and the Motor Tacho unit are fed through the cables terminating in octal plugs fitted to both Motor Tacho and Servo Amplifier Unit.

Both Power Supply Unit and Servo Amplifier Unit are fitted with sockets from which ±15V d.c. supplies can be drawn to operate all other units of the system.

The major features of the control system are shown in Figure 10. The 'input' and 'output' quantities are compared to produce an 'error' signal which actually operates the 'forward path', which is the whole of the system between the error signal and the final output. The forward path has an amplifier and motor. Amplifier operates the motor which drives the output or load through gearing.

In order to compare actual input and output quantities which give the error signal (which operates the forward path) the values must be converted to voltages. Transducers are used in the input and feedback lines for conversion.

Tachogenerator feedback is used as a compensating system that contains position control, velocity control and 'velocity feedback'. This provides a very powerful means of stabilizing systems and improving the transient response.

F. DATAMEDIA ELITE 2500 TERMINAL DESCRIPTION

The Datamedia Elite 2500 Video Terminal is a stand-along terminal containing an alphanumeric display, keyboard, storage, control logic and a synchronous/asynchronous communications interface.

The Elite 2500 can receive at data rates from 50 to 9600 band synchronous or asynchronous with a screen capacity of 1920 characters.

The Elite 2500 can store and identify 128 ASCII characters. The standard display format is 80 character line by 25 lines. Each character can be stored as a form field character or Blink field character, or both. All characters are formed on a 5 × 7 dot matrix.

The cursor, which is an underline, will identify the position on the screen. The next character received will be entered.

The DATAMEDIA Elite 2500 has special user definable "control functions" and the following device attributes:

- quiet operation
- editing plus roll mode
- 50 to 9600 band
- 80 characters per line
- no end of line hangups
- protected field
- computer derived or high light field (blink)
- addressable cursor
- added carriage time with printer transmit
- good reliability
- electronic keyboard.

V. DESIGN OF POWER CONTROL SYSTEM FOR THE GUNFIRE CONTROL SYSTEM

A. GENERAL

The Fire-Control System in this study is a shipboard double purpose weapon. The power control system for this weapon must be an electrically controlled hydromechanical drive mounted on the carriage of the gun mount, and serves to position the gun in azimuth and elevation in accordance with command signals from the fire-control computer. For the purposes of this study a modified servomechanism that was available at the NPS was utilized to simulate the gun mount. The design consideration for the real power control system would be similar but understandably, much more demanding.

Since the azimuth and elevation servo systems are essentially identical, just one servo system was designed.

B. OPERATION

The fire-control computer provides either present or future target position data, depending upon the mode of computer operation. These data are compared with positional data from the gun. If the desired and actual positions do not correspond, an error signal voltage is generated whose amplitude is proportional to the difference between the desired and actual gun position and whose phase is a measure of whether the gun lags or leads the desired position. The error signal is then amplified in a servo amplifier to a power level sufficient to drive a DC servomotor. This servomotor, in turn, is geared and linked mechanically to the gun indicator (output potentiometer and indicator in this study).

As indicated in Figure 11, voltages from the computer via DAC are supplied to the servomotor through the summing amplifier and power amplifier as 1:1 coarse data. The rotor of the servo is connected mechanically to the output gearing. The resultant error-signal voltages, after being combined with secondary signals to be discussed below, are amplified and applied to the power amplifier. This amplifier error signal serves to swing the gun to its desired position, i.e., on target.

C. DESIGN CHARACTERISTICS

In the study of a comparable system by Slaughter and Lachowski [Ref. 11] typical performance figures were given as follows:

Maximum Angular Velocity: 30 Degrees/Second

Maximum Angular Acceleration: 75 Degrees/second^2

The control loop in this study was designed to operate as close as possible to these same constraints.

The specifications on the desired closed-loop response were based on the velocity and acceleration characteristics of the gun mount and the requirement for stable operation at all time.

For example in Slaughter and Lochowski [Ref. 11] the analysis and tests conducted to determine the transfer function

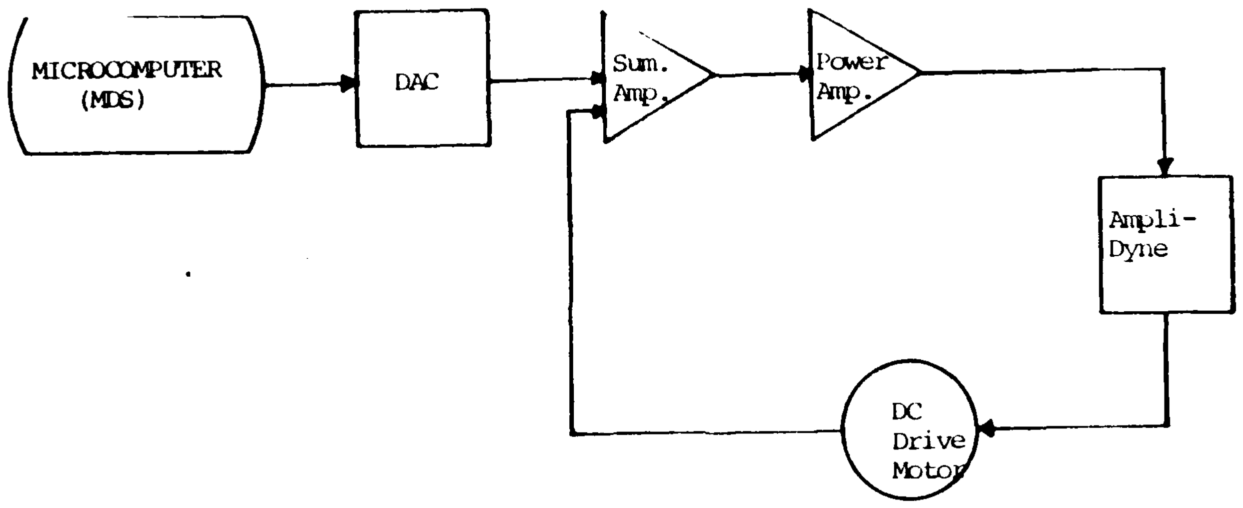


Figure 11. Power Control System Diagram for the Gun Fire Control System

of the 3"/50 gun mount, revealed that the plant to be controlled would be of the approximate form:

$$G(s) = \frac{6}{s(s + 3.5)}$$

This transfer function in a unity feedback system would have a closed loop response characterized by a damping ratio of 0.7 and a natural frequency of 2.45 radians/second. Although the damping ratio indicates that the system would surely be stable, the low natural frequency indicates that the system would be considerably more sluggish than desirable.

Considering the above factors, modifications were made on the modular servo system so the test system had a transient-response as shown in Figure 12. Characterized by a damping ratio of 0.578 and a natural frequency 6.04 radians/second.

D. DESIGN OF CLOSED LOOP SYSTEM

A closed-loop control system is one in which the output signal has a direct effect upon the control action. That is, closed-loop control systems are feedback control systems. The actuating error signal, which is the difference between the input signal and the feedback signal (which may be the output signal or a function of the output signal and its derivatives), is fed to the controller so as to reduce the error and bring the output of the system to a desired value. In other words, the term "Closed Loop" implies the use of feedback action in order to reduce system error.

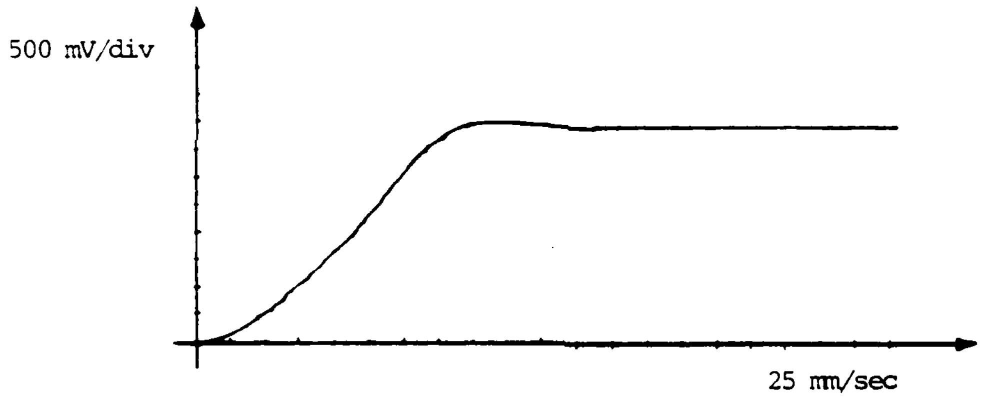


Figure 12. System Transient-Response

The major features of any simple automatic control system were shown in Figure 10.

The 'input' and 'output' quantities are compared to produce an 'error' signal which actually operates the 'forward path'. The forward path is the whole of the system between the error signal and the final output. In an electrical system, for example, the forward path may take the form of an amplifier operating a motor which drives the output as a load through gearing.

In many cases it is necessary to use some form of transducer in the input and feedback lines in order to convert the actual input and output quantities (which may be shaft angle or some other variable) into a suitable form, commonly voltages, which may then be compared to yield the error signal which operates the forward path.

For the control system in this study one of the transducers in the feedback line is a tachogenerator providing a voltage proportional to the output shaft speed, which in conjunction with a position feedback signal is compared with a desired position represented by an input voltage.

The portion of the system used to compare input and output and produce an error is commonly termed the 'error channel'. It is important to note that there is a 'closed loop' in the system from the error through the forward path and back through the output transducer to the comparison unit. It is the effect of this closed loop which gives feedback systems their particular properties both advantageous and disadvantageous.

The great advantage of a feedback or closed-loop system is that since it is 'error-operated' it contains the facility to compensate for any departure of the output from the required condition set by the input since this departure changes the error causing a correcting signal to be applied to the forward path. Whether the departure is caused by external loading, or internal disturbance in the system, or some change in the system parameters (an amplifier gain for instance), the system compensates for the change to an extent depending on the detailed design. The disadvantage of a feedback system is that due to the closed-loop path the system may tend to give an oscillatory response to any change of input which may take some time to die away, or that the system may even become unstable and maintain self-oscillation.

The general design problem of closed-loop systems is to exploit to the full the advantages that feedback provides but to avoid the disadvantages.

1. Motor Characteristics

An essential feature of any electrical position or speed-control system is an electric motor with an associated power supply and amplifier stage to control power input to the motor in response to a control signal. In this study the amplifier stage is a servo amplifier containing power transistors. It is usually necessary for the motor to be reversible and there are two common arrangements for a motor with separate field windings. The arrangements are armature connection and field connection as in Figures 13 and 14.

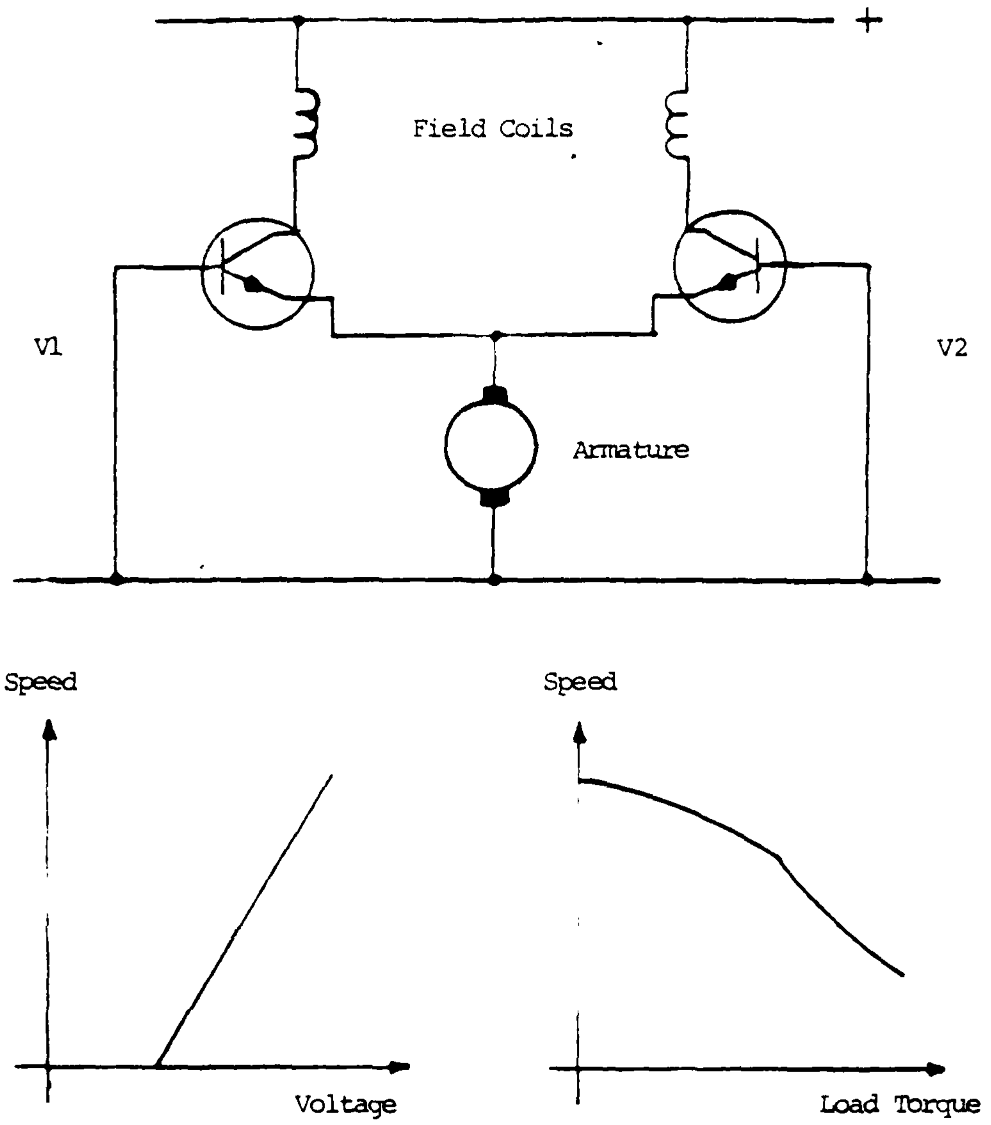


Figure 13. Armature Connection and Characteristics

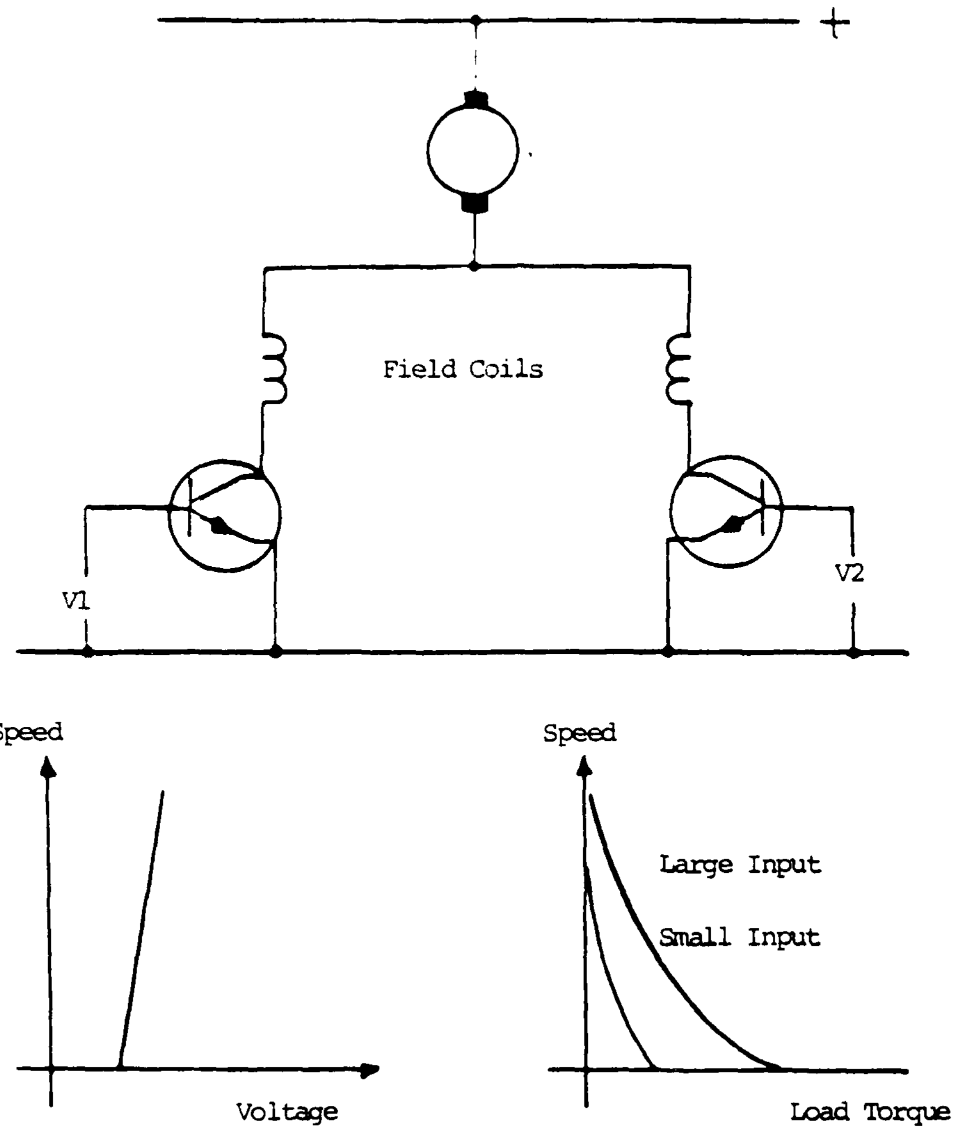


Figure 14. Field Connection and Characteristics

In the armature connection the armature is connected in the emitter circuit with one field winding in each collector, while in the field connection the armature is connected in the collector circuit. In both cases if a positive voltage (in the case of n-p-n transistors as shown in Figure 13) is applied at either V_1 or V_2, current will flow through one field and the armature, causing the motor to rotate. The two fields are so connected that the motor runs in opposite directions for V_1 or V_2. Each arrangement gives different characteristics and has particular advantages and disadvantages.

In armature connection, the fact that the armature back emf appears between the emitter and ground requires that that control voltage V_1 or V_2 must be increased to increase the motor speed, and if there is no load on the motor the speed is almost directly controlled by the input.

If the motor is loaded, the speed falls and the current increases if V_1 is kept constant. Hence the torque can rise to keep the load moving. This gives speed/voltage and speed/torque characteristics of the general form shown in Figure 13, a certain minimum voltage being required to overcome friction and make the motor rotate.

In the field connection, the transistor current is largely determined by the input signal. Hence when the minimum value required to rotate the motor is reached, and if the motor is unloaded, the speed runs up to a high value for a very small increase in input. This makes the motor difficult to control. Also if the motor is loaded the speed falls sharply. Field connection is shown in Figure 14. In general terms even though the field connection is more sensitive (has a higher 'gain') which appears advantageous, when used in a closed loop system it seriously decreases stability of the system. For this reason the Armature Control Connection was selected in this study.

Transient response for an ideal armature-controlled motor (one in which the constant brush friction is negligible compared with the torque generated by the motor), with a separately energized field, the motor speed responds exponentially to a step of input. The relation is of the same form and the response of the same shape as that of an RC circuit to a step voltage.

2. Speed Control

The units were connected to give the speed control system shown in Figure 15, in which the tachogenerator voltage is effectively subtracted from a reference voltage to give an error signal which finally drives the motor.

Ideally if the forward path gain is very high, only a small error signal is required to operate the motor and then the motor speed will be such that the tacho voltage substantially equals the reference voltage, so that the speed is controlled by the reference voltage. The steady state operating conditions of an ideal system may be represented by the following relations.

$$\dot{\theta}_o = K\,E$$

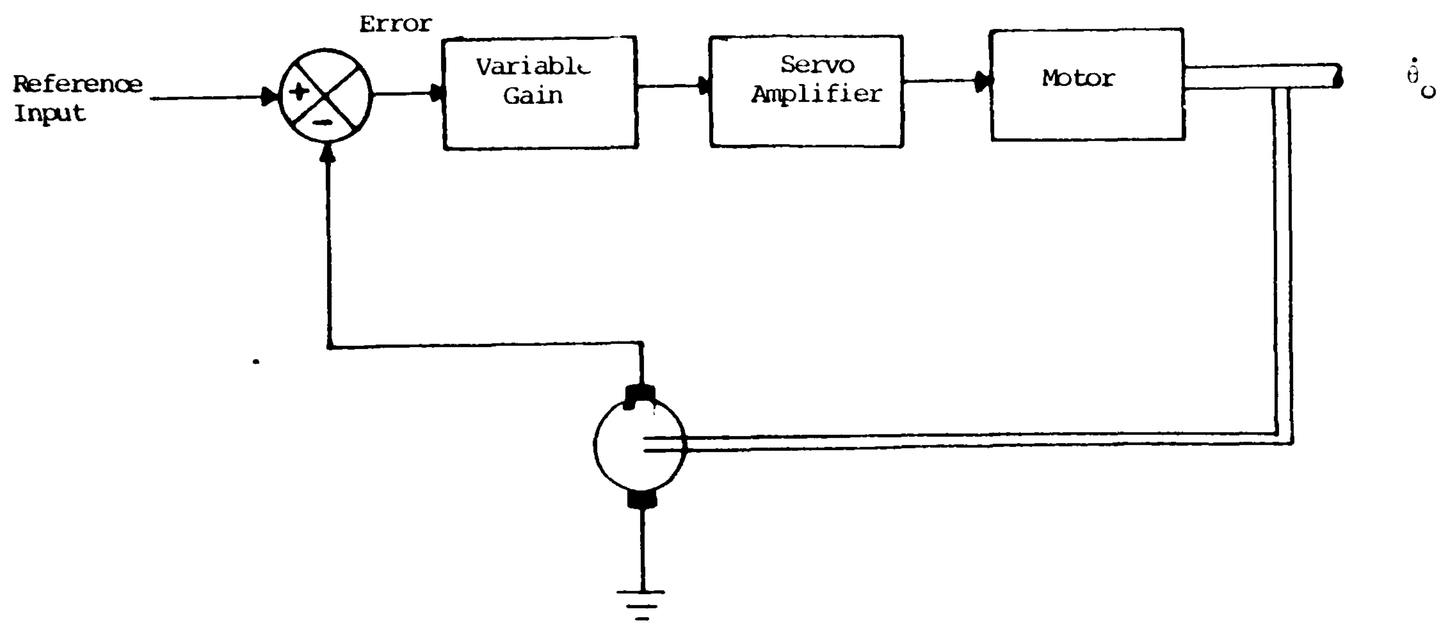


Figure 15. Simple Speed Control System

i.e., speed is proportional to error signal and K is the 'gain' of the system. Also

$$E = Vref - K_g \dot{\theta}_o$$

i.e., error is the difference between the reference voltage and the generated voltage. These two relations may be combined to eliminate the error E, giving: $\dot{\theta}_o = K(Vref - K_g \dot{\theta}_o)$ by algebraic manipulations, speed is given as

$$\dot{\theta}_o = \frac{K\ Vref}{1 + K\ K_g}$$

Hence if the forward path gain is large, this last relation becomes very nearly:

$$\dot{\theta}_o = \frac{Vref}{K_g}$$

but as $K_g \doteq$ constant,

$$\therefore \quad \dot{\theta}_o \propto Vref$$

and the speed is directly proportional to the reference voltage, the relation depending only on the tachogenerator constant which will not vary significantly. The speed reference voltage relation will not be affected by 'gun' variations provided that the gain remains high. This of course is the advantage of using a feedback system.

In the experimental setup the operational amplifier unit was used to combine the functions of a comparison unit and the variable gain stage. The output of the operational amplifier was used to drive the servo amplifier which requires a positive input to rotate the motor in a "positive" direction and a negative signal for "reverse" rotation.

In order to obtain a Reversible Speed Control it is necessary that when the reference voltage is reversed a drive signal is applied to the other transistor input in the servo amplifier. This cannot be obtained with the operational amplifier unit but a suitable drive is provided by the pre-amplifier unit.

The pre-amplifier unit has two outputs and a positive input (to either input) gives a positive output at one output, and a negative input gives a positive output at the other socket. Therefore if this unit is used to drive both inputs in the servo amplifier the motor can be driven in both directions.

3. Position Control

In a position control system a common requirement is for a motor to rotate an output shaft through the same angle as an input shaft. In this study, the inputs are the predicted bearing and elevation values.

The general form of the block diagram of a position control system is in Figure 16, but for a position control the input and output "transducers" must measure input and output shaft angles and produce a control signal (or 'error') proportional to the angle between the shafts (in this case difference between input value and output value).

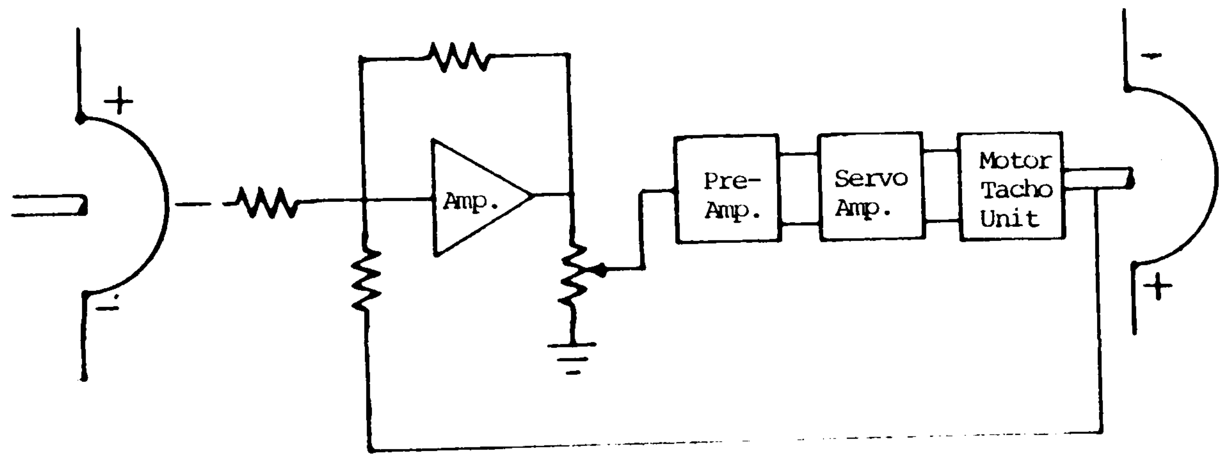


Figure 16. Simple Position Control Diagram

The major problems of position control are 'Reduction of Overshoot and Settling Time'. The high gain is desirable to reduce the deadband, but leads to increased overshoot. The system that is needed is some arrangement to enable high gain to be used without causing too serious overshoot.

The method of preventing overshoot involves using an additional control signal proportional to the velocity of the output shaft which can be obtained from the tacho generator. This is called velocity feedback, and is very often used in practice to improve system performance, as shown in Figure 17.

The real cause of overshoot is that the drive to the motor does not reverse (hence tending to slow the motor down) until the system reaches alignment (which is really too late). If the motor drive could be reversed before the system aligns, then the motor would be slowing up as alignment is approached and the overshoot would be greatly reduced. This is the effect that is obtained by 'Velocity feedback'.

The tacho generator measures the speed of the output shaft which may be increasing as the system moves towards alignment. If this signal is arranged to be negative from the tachogenerator it then is added to the error.

The sum of the signals becomes zero and reverses before the system aligns and hence the motor drive reverses before alignment. This greatly reduces the overshoot but excessive velocity feedback gives a very slow response.

Introducing velocity feedback stops the oscillation because it modifies the speed transfer characteristics

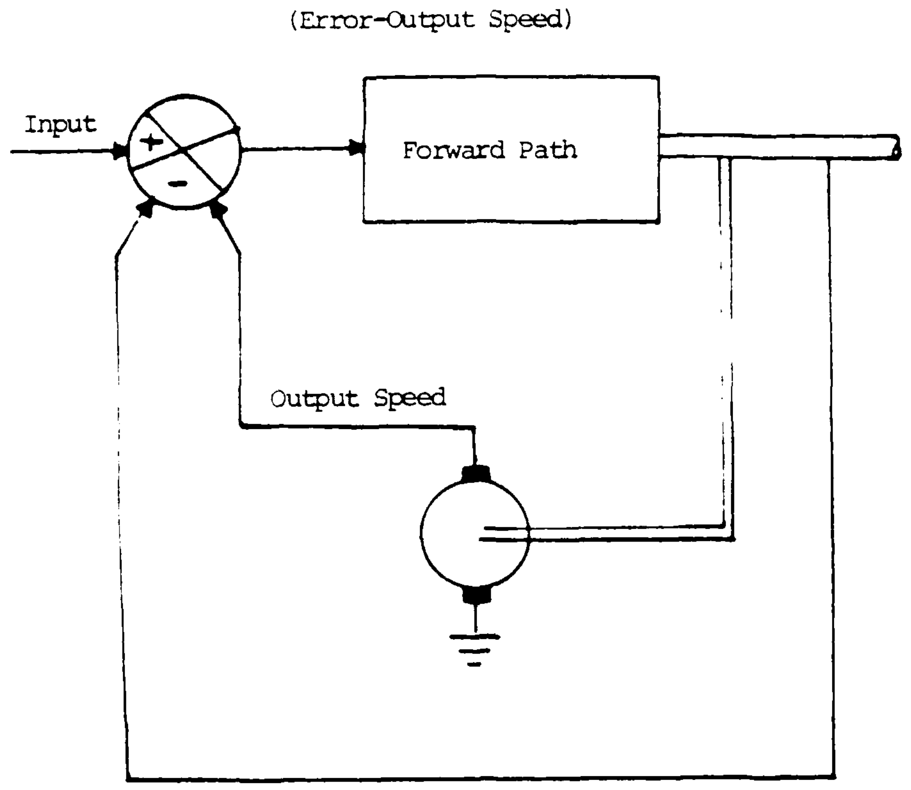


Figure 17. Velocity Feedback Block Diagram

between the input to the pre-amplifier and the motor shaft.

4. Analysis of System

Normally introductory theoretical analysis is developed assuming that motors are 'linear', in particular that frictional characteristics are viscous. In actual practice for small motors the brush friction (which is substantially independent of speed) is the dominant frictional effect, and conventional linear analysis is not applicable.

In order to enable a limited range of numerical experiments to be carried out to provide an introduction to the analytic theory of control systems, it is possible to introduce a compensating network into the pre-amplifier which in conjunction with tachogenerator feedback provides an overall motor characteristic which approximates a linear motor with a single time constant. This is referred to as a 'defined time constant' system.

Initially the general theory of simple control systems is developed and this is followed by a series of experiments investigating such points as

- Motor time constant
- Closed-loop transient and frequency response
- Instability
- Tachogenerator feedback.

5. Analysis of Simple Position Control

In order to obtain an analytic expression of the system, it is necessary initially to obtain equations to describe the performance of an idealized motor. The most convenient way to analyze the motor is to use the torque as the link between the electrical and mechanical portions, and the following equations can be written where it is assumed that the various parameters are in a compatible system of units. The motor torque is proportional to the armature current, hence

$$T = Kt\ Ia$$

where T = torque, Kt = torque constant (torque/amp.). The armature current depends on the difference between the applied voltage Vs and the generated back emf KbS.

$$Ia = \frac{Vs - Kb\ S}{Ra}$$

where S = speed, and Kb = generated back emf constant (volts/rad/sec). Finally the motor torque is used to accelerate the total inertia of the motor and load (J) and overcome the viscous friction torque, giving

$$T = \frac{JdS}{dt} + FS$$

where F = viscous friction constant (torque/rad/sec).

These equations may be combined to eliminate the torque and armature current finally leading to

$$\left(\frac{JRa}{FRa + KbKt}\right) \frac{dS}{dt} + S = \left(\frac{Kt}{FRa + KbKt}\right) Vs$$

This equation is identical in form with the equation from the analysis of a RC circuit. Hence the relation between applied volts and speed for the motor is the same as that which would be obtained between the input and output volts for the simple circuit. Thus if volts are suddenly applied to the motor it will run up to the corresponding maximum speed with a time constant

$$\tau_m = \frac{J\ R_a}{F\ R_a + Kb \cdot Kt}$$

The factor $K_t/(FR_a + KbKt)$ indicates how fast the motor will run per volt applied at d.c. and is termed the 'speed constant' Ks.

$$Ks = \frac{Kt}{FR_a + Kb \cdot Kt}$$

The frequency response relation between supply volts and motor speed is given by

$$\frac{S}{Vs}(J\omega) = \frac{Ks}{(1 + J\omega\tau_m)}$$

For control systems where the motor normally rotates an output shaft to a desired angle, it is necessary to establish a relation between applied volts and actual shaft position angle. The position or total motor shaft angle θ_m, is given by the integral of the speed

$$\theta_m = \int S\ dt$$

since $d\theta_m/dt = S$. Considering the exponential form of signals

$$\int e^{j\omega t}\,dt = \frac{1}{j\omega}e^{j\omega t}$$

so that the total shaft angle is given by

$$\frac{\theta_m}{Vs}(j\omega) = \frac{Ks}{j\omega(1 + j\omega\tau_m)}$$

For any closed-loop system the error signal is used to operate the forward path, the error being the difference between the input and output shaft angles,

$$E = \theta_i - \theta_o$$

assuming that the error, which will be in radians, is converted to a voltage and subsequently amplified so that one radian gives Kg volts, then

$$Vs = E \cdot Kg$$

If the frequency response of the system is being considered, and using the relation previously obtained for the motor shaft angle, then

$$\theta_m = \frac{Kg \cdot Ks}{j\omega(1 + j\omega\tau_m)}E$$

Since there is a gear reduction 1:N to the output shaft, then

$$\theta_o = \theta_m/N$$

leading to

$$\theta_o = \frac{Kg \cdot Ks/N}{j\omega(1 + j\omega\tau_m)} E$$

This may be substituted into the error relation to give finally the very important result

$$\frac{\theta_o}{\theta_i}(j\omega) = \frac{\dfrac{Kv}{j\omega(1 + j\omega\tau_m)}}{1 + \dfrac{Kv}{j\omega(1 + j\omega\tau_m)}} = \frac{Kv}{j\omega(1 + j\omega\tau_m) + Kv}$$

where

$$Kv = \frac{Kg \cdot Ks}{N}$$

The factor Kv determines how fast the output shaft rotates for a constant error and is called the 'velocity error constant', and is an important parameter in controlling the steady state following accuracy of the system.

Although the closed-loop expression has been developed in terms of frequency response it is also possible to establish a differential equation relating the input and output under closed-loop conditions. The differential equation governing the motor speed is

$$\tau_m \frac{dS}{dt} + S = Ks.Vs$$

but since speed (S) is the differential of position, hence

$$S = \frac{N \cdot d\theta_o}{dt}, \qquad \text{and} \qquad \frac{dS}{dt} = \frac{Nd^2\theta_o}{dt^2}$$

where θ_o is the output shaft angle, and N the reduction ratio.

Also

$$Vs = Ke \cdot Kamp$$

and these may be substituted into the motor equation to give

$$\tau_m \frac{d^2\theta_o}{dt^2} + \frac{d\theta_o}{dt} = Kv\ E$$

since Kv = (Ke Kamp Ks)/N. Finally since

$$E = \theta_i - \theta_o$$

the equation may be written as

$$\tau_m \frac{d^2\theta_o}{dt^2} + \frac{d\theta_o}{dt} + Kv\ \theta_o = Kv\ \theta_i$$

which is a second order differential equation. If this equation is compared with the 'normalized' form of a second order equation

$$\frac{d^2y}{dt^2} + 2\ \xi\ \omega_n \frac{dy}{dt} + \omega_n^2\ y = \omega_n^2\ x$$

it can be seen that the parameers ω_n, the undamped natural frequency, and ξ, the damping factor are given by

$$\omega_n = \sqrt{Kv/\tau_m}\ ; \qquad \xi = \frac{1}{\sqrt{Kv\tau_m}}$$

The frequency response corresponding to the normalized equation is given by

$$\frac{Y}{x}(j\omega) = \frac{1}{(j\frac{\omega}{\omega_n})^2 + 2\xi\frac{j\omega}{\omega_n} + 1}$$

and a phase shaft of 90° occurs when $\omega = \omega_n$.

6. Tachogenerator Feedback Compensation

In the two systems considered previously there was a direct connection between Kv and overshoot, and it was not possible to increase Kv without increasing the overshoot. Also the speed of response could not be increased since this is controlled by the 'resonant frequency' which is determined by the internal parameters of the system. In order to be able to adjust system parameters to improve the performance, one method is to introduce additional passive networks in the forward path. This is termed 'compensation'.

Another method of compensating a system is to add a signal from a tachogenerator into the forward path. Since this provides a very powerful means of stabilizing systems and improving the transient response this method was selected for compensating the system in this study. It is often termed 'velocity feedback' because the signal used is proportional to the output shaft speed.

The variable tachogenerator feedback was connected into the pre-amplifier as shown in Figure 18. The block $K_1/j\omega(1 + j\omega\tau_m)$ together with the addition unit immediately preceeding it represents the complete transfer function of the defined time constant forward path from the input to the pre-amplifier to the motor shaft. The signal E_3 only exists internally

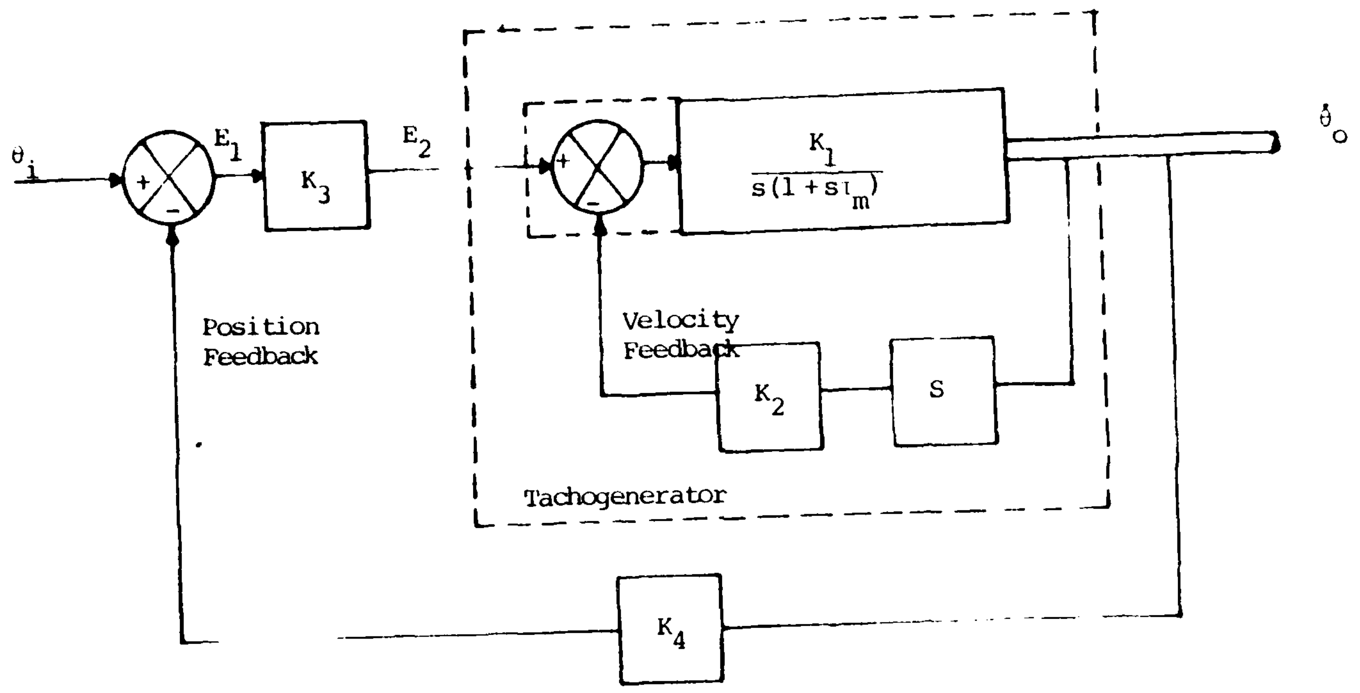


Figure 18. Tachogenerator Feedback System Diagram

in the pre-amplifier and can not be examined. The signal shown from the tachogenerator through K_2 represents a separate signal fed to the pre-amplifier input. The negative sign associated with the tachogenerator input arises automatically from the tachogenerator polarity when the defined time constant arrangement is used.

The tachogenerator actually measures the differential of the output shaft position, hence from the frequency response point of view considering exponential signals of the form $e^{j\omega t}$, the transfer of the tachogenerator is '$j\omega$', since

$$\frac{de^{j\omega t}}{dt} = j e^{j\omega t}$$

The variable factor K_2 represents a potentiometer across the tachogenerator output, and K_3 is the other potentiometer used as the normal gain control. The operational amplifier compares input and output to generate the error E_1, which passes through K_3 to provide the signal, E_2 which applied to the pre-amplifier. In the diagram there are two separate loops, the 'inner' loop, shown within the dotted box, through the defined time constant forward path and back through the tachogenerator, and a second or 'outer' loop from the output shaft through the operational amplifier and then through the complete transfer represented by the inner loop from E_2 to θ_o. The 'inner' loop transfer may be evaluated as:

$$\frac{\dot{\theta}_o}{E_2}(j\omega) = \frac{\dfrac{K_1}{j\omega(1 + j\omega\tau_m)}}{1 + \dfrac{K_1 K_2 j\omega}{j\omega(1 + j\omega\tau_m)}}$$

$$= \frac{K_1}{1 + K_1 K_2} \frac{1}{j\omega(1+j\omega\tau_m/1+K_1K_2)}$$

$$= \frac{K_1}{1 + K_1 K_2} \frac{1}{j\omega(1 + j\omega\tau_t)}$$

where $\tau_t = \tau_m/(1 + K_1K_2)$ so that the tachogenerator feedback has the effect of making the inner loop transfer similar to the open-loop speed control transfer function with both the time constant and the 'speed factor' K_1 (which is the maximum Kv value for the simple defined time constant arrangement) being reduced by the factor $1/(1 + K_1K_2)$, so that the system reduces to the equivalent diagram shown in Figure 19.

From the differential equation point of view the following equations can be considered to be the transfer θ_o/E_3

$$\tau_m \frac{d^2\theta_o}{dt^2} + \frac{d\theta_o}{dt} = K_1E_3$$

but

$$E_3 = (E_2 - K_2 \frac{d\theta_o}{dt})$$

and

$$E_2 = K_3(\theta_i - \theta_o).$$

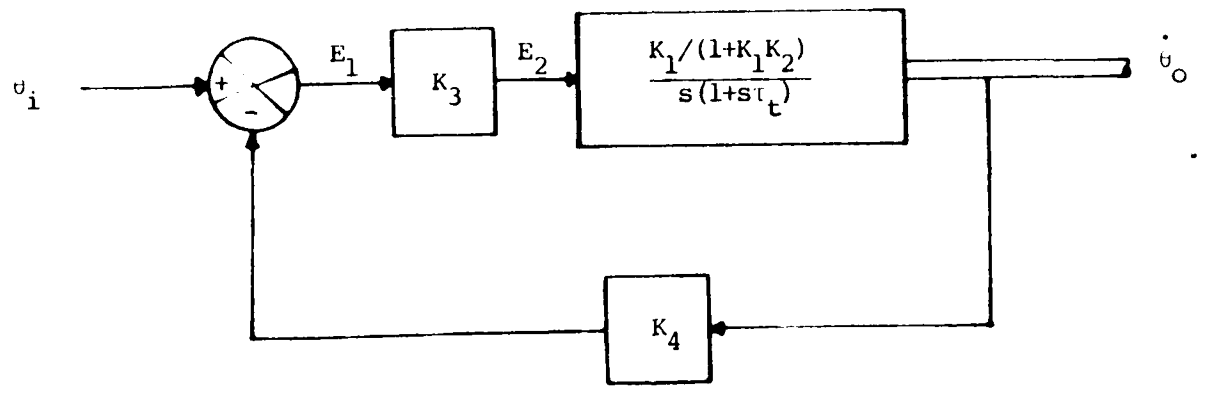


Figure 19. Tachogenerator Feedback System Reduced Form Diagram

Hence finally the overall equation can be obtained

$$\tau_m \frac{d^2\theta_o}{dt^2} + (1 + K_1K_2)\frac{d\theta_o}{dt} + K_1K_3\theta_o \doteq K_1K_3\theta_i$$

Comparing this with the normalized form of the second order differential equation gives

$$\omega_n = \sqrt{(K_1K_3)/\tau_m}$$

and

$$\zeta = \frac{1}{2}\frac{(1 + K_1K_2)}{\sqrt{K_1K_3\tau_m}}$$

so that the natural frequency is dependent upon the forward path gain K_1K_3 and the damping factor can be controlled by K_2. Hence, in principle, the natural frequency can be increased by increasing K_1K_3 while the damping is kept constant by increasing the tachogenerator feedback K_2.

The frequency response expression relating the output shaft angle and error is:

$$\frac{\theta_o}{E}(j\omega) = \frac{Kv}{j\omega(1 + j\omega\tau_m)}$$

where Kv is the velocity error constant. This gives the output shaft speed per unit error in the steady state. (The dimensions of Kv being

$$\frac{\text{radians/second at the output shaft}}{\text{radian error in volts}}\text{ .)}$$

This is an important factor in determining the steady state and also affects the magnitude of the open-loop frequency response locus and hence controls the magnitude of the peak in the closed-loop step response.

If in principle the input shaft is rotated at a constant velocity, the system will finally settle with the output shaft rotating at the same speed as the input shaft, but lagging behind with an angle just sufficient to provide the error signal necessary to rotate the output shaft at the same time speed as the input shaft. For a given input shaft speed $\dot{\theta}_i$, the following error is given by

$$E = \frac{\dot{\theta}_i}{Kv}$$

Increasing Kv reduces the steady state following error, but increases the oscillatory nature of the transient response so that for any simple system a compromise value must be chosen.

Kv is an overall factor which is the product of a number of individual factors in the forward path, in fact

$$K_v = \frac{K_e . K_{amp} . K_s}{N}$$

where:

K_e	=	volts to amplifier/radian error; called the 'error constant'
K_{amp}	=	amplifier gain
K_s	=	motor speed constant
N	=	gear ratio

In this study the value of K_v found by measurement as explained in [Ref. 12] $K_{v(max)}$ is given by

$$K_{v(max)} = \frac{4\pi}{\theta_r^\circ/57^\circ}$$

where θ_r° = the input potentiometer angle that was found by measurement to be 4°. Hence K_v or in closed-loop syste K, is equal to

$$K_1 = K_{v(max)} = \frac{4\pi}{4^\circ/57^\circ} = 179$$

The system time constant with load was found from the open-loop transient response to be

$$\tau_m = 3.6 \text{ sec}$$

Another system constant value that must be found is tachogenerator feedback constant K_2. The actual dimensions of K_2 are:

$$\frac{\text{radians}}{\text{(rad/sec) at the output shaft}}$$

The value of K_2 was determined experimentally as in [Ref. 12] and was found to be

$$K_2 = 0.1354559$$

The following loop transfer functions were explained above.

$$\tau_t = \frac{\tau_m}{(1 + K_1K_2)} = 0.142593 \text{ sec}$$

AD-A102 693   NAVAL POSTGRADUATE SCHOOL MONTEREY CA
MICROCOMPUTER BASED SHIP-BOARD GUN CONTROL SYSTEM.(U)   F/G 9/2
MAR 81 A ERDOGAN

UNCLASSIFIED   NL

2 OF 7
AD
A102693

Therefore the inner loop transfer function is

$$\frac{\theta_o}{E_2} = \frac{K_1/(1+K_1K_2)}{s(1+s\tau_t)} = \frac{49.72}{s(s+7.0129)}$$

The all system transfer function is

$$\frac{\theta_o(s)}{\theta_i(s)} = \frac{\dfrac{K_3K_1}{s(1+s\tau_t)}}{1+\dfrac{K_3K_1}{s(1+s\tau_t)}\cdot K_4}$$

where:

K_3 is the forward path gain, and

K_4 is the position feedback gain.

After calculation,

$$\frac{\theta_o(s)}{\theta_i(s)} = \frac{45.74}{s^2+7.01\,s+36.59}$$

VI. CONCEPTUAL DESIGN

Microprocessor technology can be used to solve Gun Fire Control, Maneuvering Board problems, to present a geographical plot of own ship and contact positions and to direct the gun to the target.

In order for such a system to be competitive with existing equipment, the following requirements should be fulfilled:

1. Speed and accuracy in performing necessary functions.
2. Reliable and widely available components.
3. Relatively inexpensive acquisition and maintenance costs.
4. Human engineered user interface.
5. Flexibility in its use.
6. Capacity, speed and accuracy of calculation of generated target values, gun orders and fuze setting value, while maintaining at the same time, a constant update of the own ship's position.
7. Capacity of displaying a geographic plot of the own ship and contacts, while maintaining at the same time, a constant update of the own ship's position.
8. Capacity to solve Maneuvering Board problems such as the determination of CPA information, and calculation of course and speed values for the contacts.
9. Operation of the proposed system should not require more people than the current methods.

It was with these objectives in mind that the microcomputer system described herein was designed.

A. HARDWARE DESIGN CONSIDERATIONS

The equipment selected to implement the system described in this thesis was chosen because it was available at the Naval Postgraduate School and was representative of commercially available microcomputers, plasma displays, ADC/DAC board and experimental feedback control servomechanism technologies. The selection of the INTEL Microcomputer Development System (MDS) computer as the prototype for this project was made because of its immediate availability at the Naval Postgraduate School and because Babin and Seaman [Ref. 1] had already interfaced the MDS with a plasma display, Goncalves and Bravo [Ref. 2] had implemented 'Surface-Subsurface Contact Plotter System' and Mariatequi and Hall [Ref. 3] had implemented 'Interactive Display System'.

Two features of the MDS system that were especially useful were: (1) The 8-level, nested interrupt priority resolution network, and (2) the real-time clock logic, used to maintain a real time clock value by means of generating an interrupt at 0.77 millisecond intervals.

The Feedback Modular Servo System was chosen because it is useful for experimental purposes and the only servomechanism available at NPS. The design of a Power Control System for Fire Control System is another thesis topic and is beyond the scope of this thesis.

A Datamedia Elite 2500 Video Terminal was chosen to present alphanumeric information, because its features included: (1) editing and roll operation modes, (2) 50 to 9600 band programmable speed transmission, (3) protected fields, (4) computer derived high light field (blink), (5) addressable cursor, and (6) provision to drive up to 16 external monitors.

Analog Input/Output System was chosen to interface between the sensor to SBC and MDS to servomechanism, because it provides two functions that interface directly to Intel's SBC 80 and Intellec MDS microcomputers. The functions are: (1) Analog Data Acquisition and (2) Analog Output. The device is electrically and mechanically compatible with any SBC 80 and Intellec MDS and available at NPS.

Because of the volume of floating point calculations required, an SBC 310 High Speed Mathematics Unit, developed by Intel Corporation, was incorporated. In performing high speed mathematical functions, the Math unit acts as an intelligent processor, performing a repertoire of up to 14 arithmetic functions at least an order of magnitude faster than comparable software routines.

1. <u>Simulators--Analog to Digital Interface</u>

Sensor (radar simulator) and target simulator were interfaced to the ADC. The Analog Input Output system has 8 to 32 analog inputs and six of them were used in this project.

The target simulator has three potentiometers. They generate target velocity component values (V_x, V_y and V_z). As shown in Figure 20, target simulator potentiometers connected

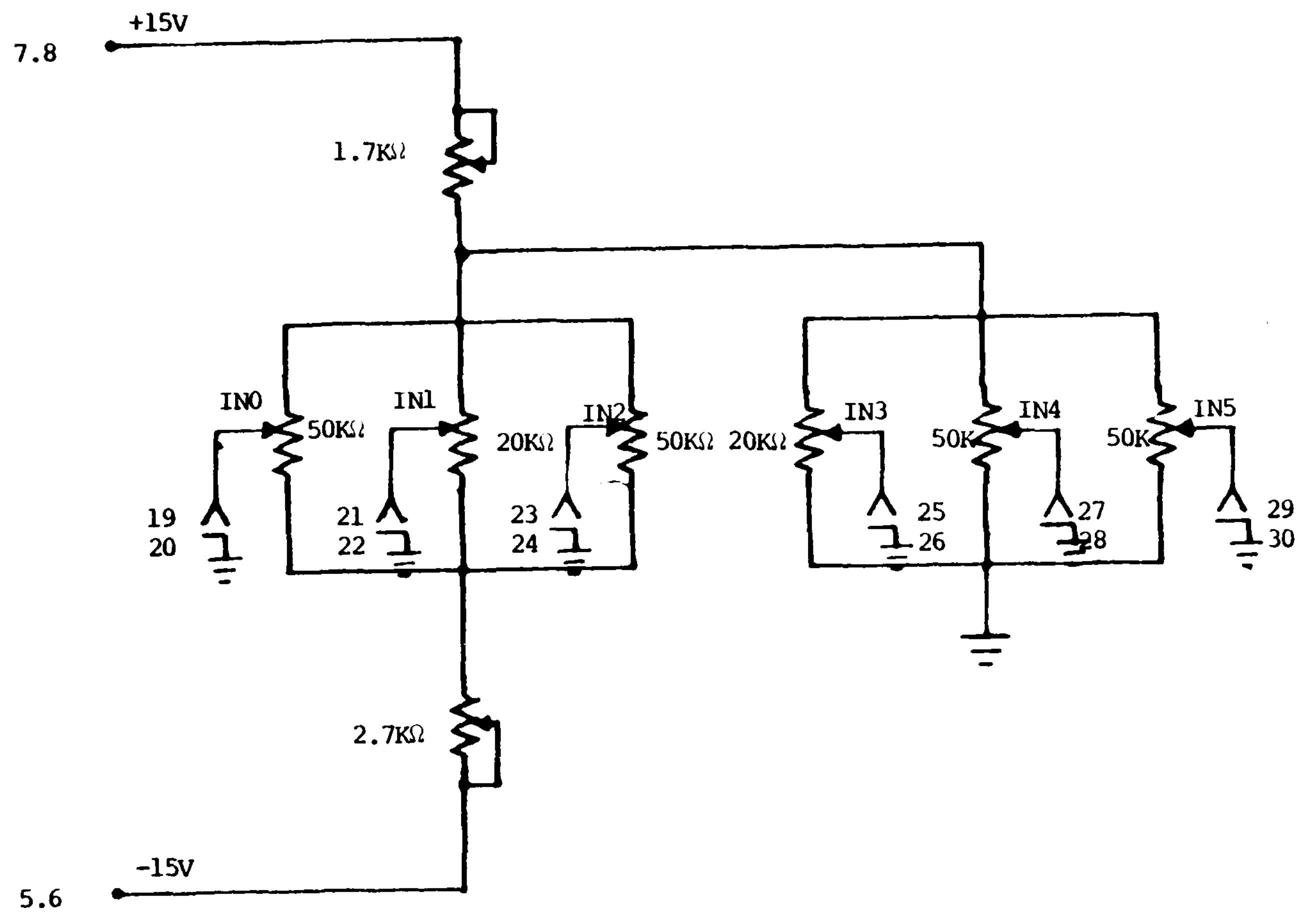


Figure 20. Simulators--ADC/DAC Interfact Circuit Diagram

to ADC board are associated with memory locations F700H, F701H and F702H. These inputs are scaled in the software program so that the 10 turn dials can simulate target speed and course.

The sensor simulator has three potentiometers also. The potentiometers are uni-directional because range, bearing and elevation have values from 0-99999, 0-360° and 0-90°. These inputs are scaled by software so operator can apply any value in the above limits. The hardware connections are shown in Figure 20. Sensor inputs are attached to memory locations F703H, F704H and F705H.

The ADC.DAC board has a direct interface to Intel's SBC 80/20.

2. Analog to Digital--Digital to Analog Interface

The Analog Input/Output System (ADC/DAC) is a self-contained unit on a circuit board which functions on the system's bus (MULTIBUS) in the same fashion as the system's memory. Because of this, a 16K byte portion of memory must be removed in order to use the ADC and DAC in the development system. Since the computer programs of the system require more than 48K bytes of memory, the loss of 16K bytes of memory is intolerable. Therefore, the ADC and DAC interface is implemented as a separate unit controlled by a single board computer, SBC 80/20.

The ADC and DAC board is placed into a separate framework which contains its own power supply, its own system's bus, the MULTIBUS, and the controlling single board computer, SBC 80/20. This independently operating subsystem is connected to the MDS developmental system with a parallel port interface. A functionally identical SBC 80/20 is resident in the developmental system which serves as a receiver and transmitter of information between the ADC and DAC and the memory of MDS. The functional diagram in Figure 21 illustrates how the systems are configured.

a. Functional Description of the ADC-DAC Interface

The single board computer which controls the ADC and DAC cyclically samples the system's analog to digital channels, located at adsolute memory locations F700H-F70FH. Each memory mapped ADC channel responds in 50 microseconds, just as a slow memory device would. The sampled analog values in the sixteen channels are placed into a buffer area for transmission to MDS.

After all sixteen channels are sampled, the values are transmitted to the SBC 80/20 in the MDS system, which then transmits the two values to be sent to the DAC for feedback control. The SBC 80/20 which controls the ADC-DAC places the two received values into a temporary buffer and forwards the values to the DAC.

The synchronization between the processors is accomplished by the use of control variables and control ports which transmit control variables. Two primitive operations, SIGNAL(CV1,RDY) and WAIT(CV1) permit synchronization between processes. SIGNAL(CV1,RDY) causes the control variable CV1, which is associated with an event (such as a

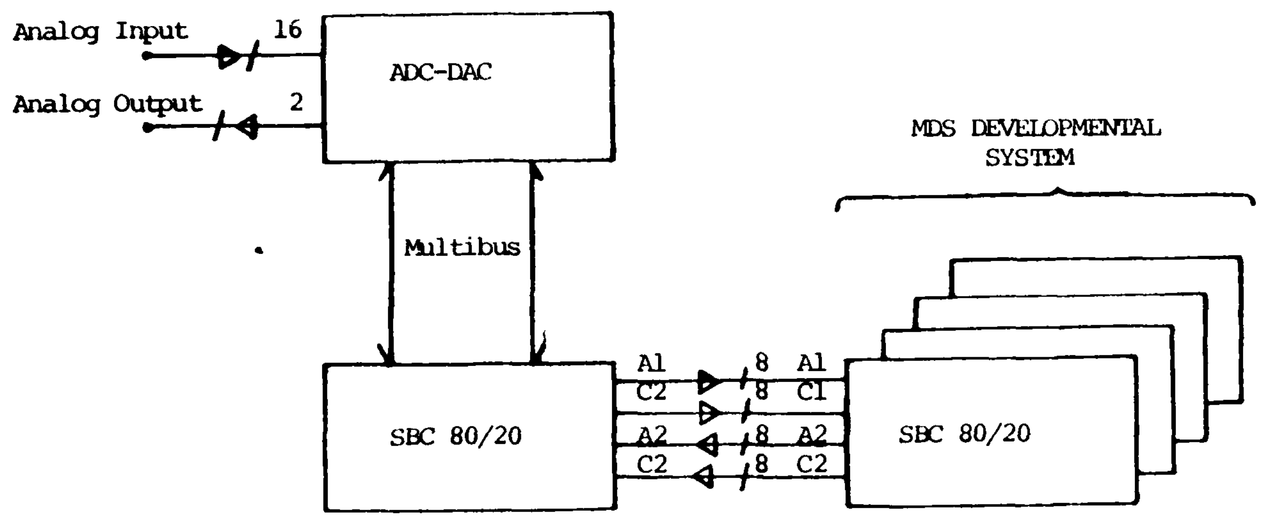


Figure 21. Functional Diagram of the Parallel Port SBC to MDS Interface

completion of a procedure) which indicates that the procedure has been completed and the results of the procedure are available for use. SIGNAL(CVI,NRDY) would reset the control variable to a not-ready status. WAIT(CV1) causes the program to wait until control variable CV1 assumes the ready status.

The following PL/M code sequence illustrates how sixteen values are transmitted and received over the parallel ports using the SIGNAL and WAIT primitives. The complete programs for ADC-DAC interface appear in Appendix G.

```
  Transmitter:
DO I = 0 TO 15;
    OUTPUT(DATAPORTA1) = BUFFER(I);/* PLACE DATA VALUE INTO
                                      DATAPORT 1 */
    CALL SIGNAL(CP1,RDY); /* PLACE READY SIGNAL IN CONTROL PORT 1*/
    CALL WAIT(CP2); /*WAIT FOR ACKNOWLEDGE OF RECEIPT OF
                      DATA ON CONTROL PORT2 */
    CALL SIGNAL(CP1,NRDY); /*DATA VALUE NOT READY IN DATA
                             PORT */
END;

  Receiver:
DO I = 0 TO 15;
    CALL WAIT (CP1); /*WAIT FOR SIGNAL WHICH INDICATES
                       THAT DATAPORT CONTAINS DATA*/
    BUFFER 1(I) = INPUT(DATAPORT A1); /*GET DATA VALUE*/
    CALL SIGNAL(CP2,RDY); /* ACKNOWLEDGE RECEIPT OF DATA */
    CALL DELAY; /* ALLOW SUFFICIENT TIME FOR SIGNAL CP2
                   READY TO BE */
    CALL SIGNAL (CP2,NRDY) /* PICKED UP */
END;
```

The parallel ports are programmable to operate in one of three modes. The operating mode and the programming is described in [Ref. 10].

In this application, the single board computers have convenient pin locations on top of the board. The two boards,one in the MDS developmental system and the other in its own enclosure are connected by flat cables. Port A1 and Port A2 are used for data transfer and Port C1 and C2 are used as control information ports. Port A1 and C1 on the SBC 80/20 in the MDS system are input ports whereas A2 and C2 are output ports. The opposite is true for the ports on the SBC 80/20 in the ADC/DAC subsystem.

3. Digital to Analog--Servomechanism Interface

The DAC has a 1 ohm ouptut impedance and the servomechanism drawing only a 0.575 ma has a high input impedance, therefore they were directly connected to each other.

B. SOFTWARE DESIGN CONSIDERATIONS

1. Language Selection

Because of the hardware configuration selected and its immediate availability, PL/M 80 was chosen as the programming language to be used. PL/M 80 is a language developed by Intel Corporation and designed especially for system and applications programming for the Intel 8080 microprocessor. The ISIS-II disk operating system, also developed by Intel Corporation for the Intellec MDS system, has a resident PL/M 80 compiler, which was very useful during the implementation, debugging and testing phases.

2. Transcendental Functions

Three transcendental functions were used, namely: (1) Cosine of a given angle in radians, (2) Sine of a given angle in radians, and (3) Arctangent of the ratio of two input parameters. These functions were previously implemented based on the procedures in Hastings [Ref. 4], using floating point procedures. Goncalves and Bravo [Ref. 2] and Appendix A describes the algorithms used.

3. Floating Point Arithmetic

Floating point arithmetic was required because the range of numbers to be represented was large and sometimes unpredictable. The format used to represent floating point numbers was as required by the SBC 310 High Speed Mathematics Unit. Appendix F shows how this unit was actually implemented and the required software procedures that are needed to use it. The functions that were implemented and the execution times are listed in Table I.

4. User Interface

This is the aspect in which most systems fail, because of the complexity of the problem of trying to define what an "average user" is in any system, and in trying to encompass all the possible ways in which a user could react. It is therefore necessary to define boundaries in how the system is expected to interact with the user, while at the same time providing an acceptable range of variations.

One of the features that becomes indispensable when interacting with a human operator is error detection and

Operation Code	Typical Execution Time	Max Execution Time	OPERATION NAME
0	15	20	Fixed-Point Multiply (MUL)
1	26	30	Fixed-Point Divide (DIV)
E	84	100	Extended Fixed Point Div (EDIV)
2	84	100	Float.-Point Multiply (FMUL)
3	92	110	Float. Point Divide (FDIV)
4	33	75	Float. Point Add (ADD)
5	33	75	Float. Point Subtract (FSUB)
6	84	100	Float. Point Square (FSQR)
7	178	205	Float.-Point Square Root (FSQRT)
8	72	100	Fixed-to-Float. Conversion (FLTDS)
9	42	85	Float.-to-Fixed Conversion (FLXSD)
A	7	7	Float. Point Compare (FCMPR)
B	7	7	Float. Point Test (FZTST)
F	4	4	Exchange (EXCH)

Note: All time values are specified in microseconds. Listed times do not include time to pass arguments to the MATH UNIT and to read results upon completion; this is typically 20 microseconds.

TABLEI. SBC-310 High Speed Math Unit Functions and Execution Times

notification as soon as possible, and the capability of allowing the user to correct his own mistakes either when so told by the system, or when he or she discovers the mistake by himself. Special care was taken in this aspect, as is explained below.

One decision that was taken during system design was to reduce to a minimum the number of keys that the operator would have to press when entering data. This implied that no special character, such as carriage return, would be necessary to mark the end of an input, but this also meant at the same time that the input would be done in a pre-formatted manner. The design choice of doing all inputs in a pre-formatted way, was reinforced by the fact that personnel in a CIC team are used to communicate with each other using a pre-defined terminology. That is, if the value of the course of a contact is 090 degrees, then when communicating this value, the word "ninety" is not used but instead the message "zero", "nine", "zero" is given.

5. Levels of Correctness

As was established before, it is very important for a system that depends almost completely on the correctness of the input values that it receives, to ensure that all the input values received fall into an acceptable range of variation, and within logical and plausible limits; furthermore, the capability of error correction should also be embedded in the system.

To achieve all of this, the system was designed to have up to six levels of error detection and correction.

a. The first level is established by checking for invalid requests from the user, such as asking for information about a contact when no contacts are in the system; if this error occurs, a warning message is issued.

b. The second level of error detection is done by checking each character received from the keyboard, to determine if it has some meaning to the system. An example of this would be when trying to input an alphabetic character in a numeric field; if an error of this type is detected, then the CRT's alarm will sound and the character will not be echoed at the display.

c. The third level is established by checking a syntactically correct input against established boundaries for the type of input being expected. An example of this would be when a value of 25 is received when requesting an hours value; since 25 lies outside the boundaries established for hours ($0 \leq$ hours < 24). Thus, a warning message will be issued, the CRT's alarm will sound the value will not be accepted nor processed by the system, and the cursor will be placed at the beginning of the incorrect value; the warning message will remain on the screen until the mistake is corrected.

d. The fourth level occurs after all input has been received from the keyboard; by giving the user the chance to correct any value just input.

e. The fifth level is obtained by making the system do all the necessary prompting in a pre-determined format. This eliminated the problems that occur when more or less data than necessary is provided to the system.

f. The sixth level is carried out by allowing the user to change any current own ship or contact value at any time.

All these levels of correctness are ensured any time an input operation takes place. It is important to note that no provision for values that are syntactically correct and within established boundaries, but incompatible with previous data and logically incorrect with the present situation, could be provided because of the many parameters, variables and special situations that could be invalid at any given time.

Obviously, the use of so many checking procedures added a considerable amount of overhead to the system, as far as the amount of code is concerned; however, sufficient care was available, and execution speed was not significantly degraded.

VII. SOFTWARE OVERVIEW

A. GENERAL INFORMATION

The software application package for the system utilized "PL M-80" high level language in the MDS Microcomputer [Refs. 5, 6, 7, 8 and 9].

The software effort for the MDS microcomputer required 70 percent (%) of the project development time for completion. The software package consisted of 15 major modules, 315235 bytes of source code and 107385 bytes of object code.

B. DATA STRUCTURES

The main data structures used in the system can be classified in three categories: (1) Data Structure used to represent system parameters, (2) Data Structures used to represent the own ship information, and (3) Data Structures used to represent contacts information. The main type of data structure used was the STRUCTURE, as provided by PL/M-80 [Ref. 8], which basically allows the programmer, by using one identifier, to refer to a collection of STRUCTURE MEMBERS which may have different types, such as ARRAY, ADDRESS variables and BYTE variables. In PL/M-80, a BYTE variable is an 8-bit value occupying a single byte of storage, an ADDRESS variable is a 16-bit value occupying two consecutive bytes of storage, and an ARRAY is a vector composed of BYTE or ADDRESS variables. The most sophisticated data structure that PL/M-80 supports directly is the array of structures with Arrays inside the structures. With the capability that PL/M-80 has of allowing a variable reference to be either fully qualified, partially qualified, unqualified (references to entire arrays or structures), or by using pointers and indirect references, it was possible to simulate other more complex data structures such as circular linked lists and plexes, as will be explained later.

1. Data Structure for the System Parameters

Figure 22a describes the data structure used to represent most of the system parameters; as can be observed, it is a structure composed of eight (8) members with the following description:

a. LAT: array of 4 bytes used to represent the floating point value of the latitude parameter used to define the Coordinate Grid Origin.

b. LONG: array of 4 used to represent the floating point value of the longitude parameter used to define the Coordinate Grid Origin.

c. SCALE: array of 4 bytes used to represent the floating point value of the graphics scale parameter.

d. WIND$DIR: array of 4 bytes used to represent the floating point value of the wind direction.

e. WIND$SPD: array of 4 bytes used to represent the floating point value of the wind speed.

f. NUM$ZONE: array of 5 bytes used to represent the ASCII characters defining the value of the Time Zone number.

(a) DATA STRUCTURE FOR SYSTEM PARAMETERS

```
DECLARE SYSTEM STRUCTURE
               (LAT (4) BYTE,
                LONG (4) BYTE,
                SCALE (4) BYTE,
                WIND$DIR (4) BYTE,
                WIND$SPD (4) BYTE,
                NUM$ZONE (5) BYTE,
                CONTACT$KIND (3) BYTE,
                NUNCTS BYTE) PUBLIC;
```

(b) DATA STRUCTURE FOR OWN SHIP PARAMETERS

```
DECLARE OWN$SHIP$INFOR STRUCTURE
                     (LAT (4) BYTE,
                      LONG (4) BYTE,
                      POINTER BYTE,
                      FLAG BYTE) PUBLIC;

DECLARE OWN$SHIP (30) STRUCTURE
                     (X (4) BYTE,
                      Y (4) BYTE,
                      TIME (3) BYTE,
                      CRS (4) BYTE,
                      SPD (4) BYTE) PUBLIC;
```

Figure 22. Data Structure

g. CONTACT$KIND: array of 3 bytes used to represent the total number of contacts in each class.

h. NUMCTS: byte variable used to represent the number of contacts at any time in the system.

It should be noticed that this data structure is designed only to maintain the parameters described any moment; a log of changes or modifications to the parameters described is not kept.

2. <u>Data Structures for Own Ship Parameters</u>

Figure 22b describes the data structures used to represent the own ship parameters; the OWN$SHIP$INFO structure is used to maintain a set of parameters for which no log of changes or modifications is maintained, and also to allow indirect access to the OWN$SHIP data structure. Its four (4) members have the following description:

a. LAT: array of 4 bytes used to represent the floating point value of the latitude parameter defining the geographical position of the own ship.

b. LONG: array of 4 bytes used to represent the floating point value of the longitude parameter defining the geographical position of the own ship.

c. POINTER: byte variable used to access the OWN$SHIP data structure, by defining its most recently used member.

d. FLAG: byte variable used to indicate whether or not the 30 members of the OWN$SHIP data structure have been accessed at least once.

The OWN$SHIP Array of Structures is used to maintain up to thirty (30) different sets of values for the own ship parameters, thus maintaining a log of the 30 (or fewer) most recent values of the own ship parameters. By using the members POINTER and FLAG of the OWN$SHIP$INFO structure, it is possible to use the OWN$SHIP data structure as a circular queue, with each of its 30 members having the following description.

e. X: array of 4 bytes used to represent the floating point value of the X parameter defining the position of the own ship in the Coordinated Grid System defined.

f. Y: array of 4 bytes used to represent the floating point value of the Y parameter defining the position of the Y parameter defining the position of the own ship in the Coordinated Grid System defined.

g. TIME: array of 3 bytes used to represent the real time clock value (hours, minutes and seconds) at which the corresponding member was current.

h. CRS: array of 4 bytes used to represnt the floating value of the course parameter.

i. SPD: array of 4 bytes used to represent the floating point value of the speed parameter.

Notice that each member of the OWN$SHIP data structure is capable of providing enough information to locate the own ship; up to 30 locations and time may be recorded.

3. Data Structures for the Contact Parameters

Figure 23 describes the data structures used to represent contact parameters. The CONTACT$INFO Array of Structures, composed of (15) members, was used to represent a set of parameters for up to 15 different contacts, and was also used to allow indirect access to the CONTACT$POSI data structure; each member of CONTACT$INFOR data structure had the following configuration:

a. DESIG: address value used to represent the decimal value of the designation used to refer to a contact.

b. TYPE: byte variable used to represent the type of the contact.

c. KIND: byte variable used to represent the class of the contact.

d. CRS$FLAG: byte variable used to indicate whether or not there was information about the course of the contact.

e. SPD$FLAG: byte variable used to indicate whether or not there was information about the speed of the contact.

f. OS$POINTER: byte variable used to indicate which of the 15 corresponding members of the CONTACT$POSI data structure was current when the own ship made its last change in course or speed; it is mainly used when solving Maneuvering Board problems.

g. POINTER: byte variable used to access the 15 corresponding members of the CONTACT$POSI data structure, by defining its most recently used member.

```
DECLARE CONTACT$INFOR (15) STRUCTURE
                      (DESIG ADDRESS,
                       TYPE BYTE,
                       KIND BYTE,
                       CRS$FLAG BYTE,
                       SPD$FLAG BYTE,
                       OS$POINTER BYTE,
                       POINTER BYTE,
                       FLAG BYTE) PUBLIC;


DECLARE CONTACT$POSI (225) STRUCTURE
                      (X (4) BYTE,
                       Y (4) BYTE,
                       TIME (3) BYTE,
                       CRS (4) BYTE,
                       SPD (4) BYTE,
                       BRG (4) BYTE,
                       RNG (4) BYTE
                       ELV (4) BYTE) PUBLIC;
```

Figure 23. Data Structures for Contact Parameters

h. FLAG: byte variable used to indicate whether or not the 15 corresponding members of the CONTACT$POSI data structure have been accessed at least once.

The CONTACT$POSI Array of Structures is used to maintain up to 15 different contacts, thus maintaining a log of the 15 (or fewer) most recent values for the parameters of each contact. Its 225 members are accessed indirectly by the POINTER element in each member of CONTACT$INFO, according to the following method:

Relative position of member in CONTACT$INFO: N. Members in CONTACT$POSI to which N is allowed to access: 15 x N up to (15 x (N+1)) - 1.

Thus, for example, the contact defined as member 0 in CONTACT$INFO, has its 15 sets of parameters represented by numbers 0 through 14 of CONTACT$POSI, and the contact defined as member 8 in CONTACT$INFO, has its 15 sets of parameters represented in members 120 through 134 of CONTACT$POSI.

Also, the 15 members of CONTACT$POSI that are allocated for each contact defined in CONTACT$INFO, are used as a circular queue, in a similar way to that described for the own ship.

Each of the 15 members has the following configuration:

h. X: array of 4 bytes used to represent the floating point value of the X parameter defining the position of the contact in the Coordinated Grid System defined.

i. TIME: array of 3 bytes used to represent the real time clock value (hours, minutes and seconds) at which the corresponding member was current.

j. CRS: array of 4 bytes used to represent the floating point value of the course parameter.

k. SPD: array of 4 bytes used to represent the floating point value of the speed parameter.

l. BRG: array of 4 bytes used to represent the floating point value of the bearing parameter.

m. RNG: array of 4 bytes used to represent the floating point value of the range parameter.

n. ELV: array of 4 bytes used to represent the floating point value of the elevation parameter.

C. SOFTWARE FUNCTIONAL DESCRIPTION

1. Modules Description

The system was developed around 15 basic modules; there is another module (EXECUTIVE) which is embedded in the module MAIN$MODULE.

As shown in Appendix G, every procedure has a comment header which explains what it performs, the parameters with its meaning, and, when proper, the usage of that procedure.

The basic idea was to encompass all functions corresponding to a level of design into one software module capable of performing all the necessary functions.

In doing so, the following modules were developed:

a. EXECUTIVE$CMDS: a module containing all the necessary procedures to link all the other modules.

b. ANAMOD: a module used to serve as executive module of the system by combining all the different functions of the already described modules.

c. COMMANDS$1: a module used to interface with the user to get requested input values.

d. DISPLAY$CMDS: a module used to display information requested by the user, on the CRT.

e. CRT$1: a module used to display all the necessary and requested values on the CRT.

f. BASICS$1: a module used to interface the system with the CRT/keyboard.

g. TIME: a module used to perform all functions dealing with time, and also to keep real time clock for the system.

h. PLASMA$PRIMITIVES: a module used to interface the system with the plasma Display Unit.

i. PLASMA$MODULE: a module used to display all the necessary graphic information in the Plasma Display unit.

j. FLOATING$POINT$1: a module used to perform all the necessary floating point operations, and to calculate the transcendental functions already described.

k. FLT$TO$ASCII: a module used to make the necessary conversions from a string of ASCII characters into floating point format, and vice versa.

l. CPA: a module used to calculate and solve all CPA values and problems.

m. GFCS$AAW: a module used to calculate and solve Anti-Air Warfare gun fire control problem.

n. GFCS$SW: a module used to calculate and solve Surface Warfare gun fire control problem.

Appendix A describes all the main algorithms used in the system while Appendix G lists all the programs that compare the system, and Appendix F contains an Operator's manual giving instructions on how to use the system.

2. Module Interaction

Due to the interaction capability between modules as allowed by the language PL/M 80 through the use of the attributes PUBLIC and EXTERNAL for the procedures, the following list was written in order to show this interaction. This list shows the modules which have procedures called by the listed module.

a. ANA$MODULE
   (1) EXECUTIVE $ COMMANDS
   (2) DISPLAY$ CMDS
   (3) COMMANDS
   (4) PLASMA$MODULE
   (5) CRT
   (6) FLTASCII
   (7) FLOATING$POINT
   (8) TIME
   (9) BASICS

b. EXECUTIVE (embedded in ANAMODULE)
   (1) ANAMODULE
   (2) EXECUTIVE$ COMMANDS

(3) DISPLAY$CMDS

(4) PLASMA$MODULE

(5) CRT$1

(6) TIME

(7) PLASMA$PRIMITIVES

(8) BASICS

c. EXECUTIVE $ COMMANDS

(1) ANAMODULE

(2) EXECUTIVE $ COMMANS

(3) CPA$MODULE

(4) COMMANDS

(5) PLASMA$MODULE

(6) CRT

(7) FLTASCII

(8) FLOATING$POINT

(9) TIME

(10) BASICS

d. CPASMODULE:

(1) EXECUTIVE $ COMMANDS

(2) CPA$MODULE

(3) COMMANDS

(4) FLOATING$POINT

e. DISPLAY $ CMDS:

(1) EXECUTIVE $CMDS

(2) CPA$MODULE

(3) DISPLAY$CMDS

(4) COMMANDS

(5) CRT

(6) FLOATING$POINT

(7) BASICS

f. COMMANDS:

(1) COMMANDS

(2) CRT

(3) FLTASCII

(4) FLOATING$POINT

(5) BASICS

g. PLASMA$MODULE
   (1) EXECUTIVE $ COMMANDS
   (2) COMMANDS
   (3) PLASMA$MODULE
   (4) CRT
   (5) FLOATING$POINT
   (6) PLASMA$PRIMITIVES
   (7) BASICS

h. CRT:
   (1) CRT
   (2) BASICS

i. FLTASCII:
   (1) FLTASCII
   (2) FLOATING$POINT

j. FLOATING$POINT:
   (1) FLOATING$POINT
   (2) BASICS

k. TIME:
   (1) COMMANDS
   (2) CRT
   (3) BASICS

l. PLASMA$PRIMITIVES
   (1) PLASMA$PRIMITIVES

m. BASICS
   (1) BASICS

n. GFCS$AAW
   (1) FLOATING $ POINT
   (2) FLTASCII
   (3) BASICS
   (4) CRT
   (5) COMMANDS

o. GFCS$SW
   (1) FLOATING$POINT
   (2) FLTASCII
   (3) BASICS

(4) CRT

(5) COMMANDS

VIII. SYSTEM DESCRIPTION

The description of the system is divided into two major areas: hardware dependencies and system characteristics.

A. HARDWARE DEPENDENCIES

Because of the hardware equipment that was selected, the following hardware dependencies exist in the current implementation of the system:

1. The system utilizes the real time clock logic as provided by the MDS system; this clock is capable of generating an interrupt of level 1, when enabled, at fixed intervals of 0.77 milliseconds. In order to be able to use this feature, a procedure named clock was implemented and defined to be of type INTERRUPT 7; this allowed the PL/M-80 compiler to create the necessary code for the interrupt vector and for the routine CLOCK. A software problem needs to be explained at this point. Since the development of this system was done under ISIS-II, which would not allow any user generated code to be located below memory location 3000H, except for code to be used in interrupts of level 1 (locations 8 through 15), then in order to override this ISIS-II inconvenience, the interrupt vector generated for CLOCK, as starting in location 56 (level 7), had to be moved to location 8 (level 1), after locating the system code in memory, and before attempting to use the real time clock; this problem is described in [Refs. 5, 8, 10] in more detail.

2. The system occupies approximately 45K bytes of physical memory for code, and approximately 17K bytes to be used for variable data; therefore, a configuration of at least 62K bytes of RAM is necessary to execute the system. Because of the memory requirement the memory mapped Analog to Digital--Digital to Analog Converter had to be implemented as a separate and independent item.

3. The system utilizes an SBC 310 High Speed Mathematics Unit to perform floating point arithmetic. Although the presence of this math unit could be avoided by replacing its functions with appropriate software routines performing the same operations using the same formats, this is not recommended because of the excessive overhead that would result, especially with regard to execution time. Appendix E explains how the math unit was actually implemented.

4. The system depends in three ways on the type of terminal used: the handshaking procedure necessary to communicate between the CPU and the terminal, the code needed to control the CRT's functions, and the general features of the DATAMEDIA Elite 2500 Video Terminal, notably the programmable roll mode, the setting of privileged fields, the capability of having an addressable cursor, and the possibility of making displayed message blank.

5. Because of the design of DC servo motor, it has an unstable zone ±15 degrees both sides of 180°. In real applications, gun has ±135 degrees limit at both sides (port and starboard).

6. Finally, the system has a hardware dependency with respect to the Plasma Display Unit selected in three respects: the handshaking procedure necessary to send characters to the Plasma Display Unit; the code used to control the Plasma Display Unit functions; and the capability of the Plasma Display Unit of working either in alphanumeric or in vector mode. The SAI Plasma Display Unit has a built-in capacity to draw solid or dashed vectors by specifying the two end points defining the vector.

B. SYSTEM CHARACTERISTICS

As previously established, the system was designed to perform basically the same functions as Gun Fire Control system and Dead-reckoning equipment, while generating target values and simultaneously solving maneuvering board problems. The system, as designed in this thesis, is capable of performing the following tasks:

1. Generate target values for air targets, solve gun fire control problem and make ballistic calculations. Calculate target predicted values and direct the gun to the predicted target position.

2. Solve the gun fire control problem for the surface targets, make ballistic calculations, direct the gun to predicted target positions.

3. Maintain as many as 30 positions of the own ship; maintain as many as 15 positions for each contact, for as many as 15 different contacts.

4. Maintain and present a geographic plot of own ship and contacts.

5. Maintain and display Surface and Air-Borne Status Board.

6. Solve the following Maneuvering Board problems:

   a. CPA information

   b. Course and speed of contacts.

7. At user's request, prompt for necessary inputs and update the displays, if applicable.

8. At user's request, display all the information that exists in the system, in a pre-established format.

9. Update automatically the position of the own ship, by using its course and speed values with a frequency specified by the user.

All the above tasks are performed by the system, either automatically or at user's request. Some parameters need to be defined during system initialization; these parameters are:

1. Time zone number.

2. Local time at which the system is started.

3. Latitude and Longitude values defining a selected geographical point to be used as center of a Coordinated Grid System used mainly for plotting purposes and as a reference to determine positions of the own ship and contacts.

4. Latitude and Longitude values defining the starting position of the own ship.

5. Initial course value of the own ship.

6. Initial speed value of the own ship.

7. Initial scale value for the Plasma Display Unit.

8. Set the target speed component values dependent on the aircraft type.

9. Set the target range, bearing, and elevation values dependent on target direction that one wanted to test.

The system also has two parameters with initial default values: (1) The Safe CPA Range set at 50 yards, and (2) the interval of time between updates of the own ship's positions is set at 180 seconds. Any parameter in the system can be changed at any time, with the exception of those parameters that define the boundaries of the system. These fixed values are:

1. Maximum range: 100.0 miles.
2. Maximum speed: 99.9 knots for surface, 1500 knots for Air targets.
3. Number of letters used to designate a contact: 2.
4. Minimum scale value: 00.25 miles/inch.
5. Maximum scale value: 25.00 miles/inch.
6. Zero defined for the system:

$$-0.0000009 \leq \text{"ZERO} \leq +0.0000009$$

(for floating point numbers only).

7. Minimum Safe CPA Range value: 50 yards.
8. Maximum Safe CPA Range value: 1000 yards.
9. Maximum Range for fire control problems is 99999 yards.
10. Maximum gun bearing limit is ±135 for both sides (port and starboard).

Table II describes the formats used for input, internal, and output representation of the values with which the system operates, as well as the units used; the conversion factors used in the system are:

1 Nautical mile	= 2025.3716 yards.
1 Degree	= 0.0174532925 radians.
1 Minute	= 0.00029089 radians.
1 Minute of long	= 1 nautical mile.
1 Minute of lat	= 1 nautical mile $\times$ Cos(LAT).
PI	= 3.141593
1 Knot	= 1 nautical mile/hour.

C. SYSTEM DISPLAYS

The system was planned to maintain three different displays: a Video Terminal presents Surface and Air Status Board and interactions with the user, Plasma Display number one presents a geographic plot of the positions of own ship and contacts and Plasma Display number 2 presents the positions of own ship and target.

1. Graphical Display Mode

The Plasma Display is used to present a geographical plot of the own ship and contact positions. It displays the geographical picture that is defined through a system-defined and user-controllable "window"; this window is used to focus on the geographical area that is of interest to the user. Two mechanisms control the "window":

VALUE	FORMAT		UNITS	
	I/O	INTERNAL	I/O	INTERNAL
TIME	xx:xx:xx	3 Bytes	hrs:min:sec	Same
COURSE	xxx.x	F.P.	Degrees	Same
SPEED	xxx.x	F.P.	Knots	Same
BEARING	xxx.x	F.P.	Degrees	Same
RANGE	xxx.x	F.P.	Miles	Same
RANGE	xxxxxx	F.P.	Yards	Miles
SCALE	xx.xx	F.P.	Miles/inch	Same
LAT.	xx:xx:.x	F.P.	Degrees:Minutes	Minutes
LONG.	xxx:xx.x	F.P.	Degrees:Minutes	Minutes
DESIG	AA	Address	ASCII Characters	decimal Value

x .... Numeric Character
F.P. .... Floating Point Representation
A .... Alphabetic Character Including Space

Table II. INPUT AND OUTPUT FORMAT

a. Scale. This defines the scale at which the plotting is desired to be presented, determining the size of the window.

b. Picture Reorientation. This specifies the window's center. Three different methods were used:

(1) Eight (8) pre-defined points in the picture.

(2) By making the last position of the own ship to be new center.

(3) By making the last position of any contact to be the new center of the picture.

The Plasma Display also presents the current value of the scale being used in its upper left corner. The positions of the own ship are marked with bright circles connected by solid vectors, while the positions of the contacts are connected by dashed vectors and marked with two different symbols: a cross if the contact is hostile or unknown, and a circle if the contact is friendly; the designation of any contact plotted at the Plasma Display is presented close to its first plotted position. Appendix A gives the algorithms used for establishing the window and for forming the picture to be presented.

2. Alphanumeric Display Mode

Figure 3 presents a picture of how the Video Terminal Display is arranged. The screen is divided into two areas; the upper portion of information about the own ship and same contacts. From this representation, the following information can be obtained at any time:

(1) Time. The time zone number and the local time maintained and automatically updated by the system every second.

(2) Own Ship. Latitude and longitude values indicating its last geographical position as determined automatically by the system at least every T time units where T is a period of time as selected by the user. Information about the course and speed is also available.

(3) System. Information about the total number of contacts in the system, classified by their corresponding class: Friendly (F), Hostile (H), and Unknown (U). Information about the mode in which the system is operating is also displayed. This will be explained in the following paragraphs.

(4) Contacts. Complete information about six contacts is displayed. The following items are provided for each contact:

(a) Designation.

(b) Type: Surface (SU), Sub-surface (SS) or Air (AA).

(c) Class: Friendly (FRI), Hostile (HOS), or Unknown (UNK),

(d) Last Mark: time, bearing and range.

(e) Course, if known.

(f) Speed, if known.

(g) CPA information, if known; time, bearing and range, or one of the following three possible messages: "time COLLISION", "SAME CRS & SPD", and "MOVING AWAY".

It should be noted that, although the system is capable of maintaining up to 15 contacts, information about only ten of them will be constantly present at the display; the user has the capability of selecting which contacts he desires to be displayed in this way, or as will be explained later, he can also obtain information about any contact (temporarily) in the lower portion of the screen.

The lower portion of the screen consists of the last eight rows and is used during Input and Display operations; it has no fixed format and if the system is not executing an Input or Display operation, it contains only the prompt (%) symbol displayed in its upper left corner.

As mentioned previously, the system operates in four different modes; these modes are: (1) Initialization, (2) Input, (3) Display and (4) Fire Control Mode. The Initialization mode is required only once at the beginning of the execution of the system, and it is indispensable for the operation of the system, as was explained previously; it can not be requested by the user once the system is operating. The Input, Display and Fire Control modes are determined at any given time. The system operates in the Input mode by default. Three modes require interaction with the user.

a. Input Mode

The system is operating in Input mode any time an Input operation is being performed. The following Input operations can be requested by the user:

(1) Modify Coordinate Grid Origin parameters; optional: latitude and longitude.

(2) Modify Own Ship Parameters; optional: latitude, longitude, course and speed.

(3) Create a Contact; required: designation, type, class, bearing, and range; optional: course and speed.

(4) Remove a contact; required: designation.

(5) Redesignate a Contact; required; old and new designations.

(6) Update a Contact; required; designation; optional: type, class, bearing, range, course and speed.

(7) Contacts to Display: select which contacts are desired to be displayed permanently in the Status Board.

(8) Time; optional: time zone value, system clock value, and time between updates.

(9) Safe CPA Range: modify the safe CPA range parameter.

(10) Wind: enter/modify wind parameters; required: direction and speed.

(11) Scale: modify the graphics scale value.

(12) Plasma Reorientation: reorient the picture displayed at the Plasma Display Unit.

All these operations are performed using the lower portion of the screen and are divided into various phases ("pages") in order to allow a better utilization of the screen; any input operation can be requested by pressing the appropriate key at the keyboard.

b. Display Mode

The system is operating in Display Mode any time a Display operation is being performed. The following Display operations can be requested by the user:

(1) Origin: Display information, about the coordinate Grid Origin parameters (latitude and longitude).

(2) Scale: Displays information about the Graphics Scale currently used by the system.

(3) Own Ship: Displays information about the following own ship parameters: latitude, longitude, X and Y values in the user defined Coordinate Grid System being used, course and speed.

(4) Contact Information: Displays information about a specified contact's parameters; requires designation; provides: type, class, number of positions maintained by the system, latitude, longitude, X and Y values in the user defined Coordinate Grid System being used, last mark's time, bearing and range, and if possible, gives information about course, speed, CPA parameters, and estimated actual position.

(5) Contacts in System: Displays information about the designations of all the contacts in the system, if any.

(6) Safe CPA Range: Displays the value of the Safe CPA Range parameter.

(7) Wind: Displays information about the wind.

(8) Display Update Time: Displays the value of the current Time Between Updates being used.

All these operations are performed using the lower portion of the screen, and are divided into various phases ("pages") in order to allow a better utilization of the screen. Any display operation can be requested by pressing the approximate key at the keyboard. Figure 3 presents a view of how the Video Terminal display looks during system operation. The arrangement of the various Input and Display keys are explained in Appendix F.

c. Fire Control Mode

This mode is included by Display mode. When user interacts with system (by type key or touch panel) secondary plasma display presents own ship and target positions. The video terminal presents target and fire control values (gun orders).

d. Other Function-Defined Keys

The keyboard has also two other special function keys:

(1) Rubout Key. Used to backspace the cursor when inputting information into the system.

(2) "Go" key. Used to advance the various "pages" in which the display operations are divided.

IX. CONCLUSION AND RECOMMENDATIONS

The microcomputer based simulated interactive gun fire control display system partially implemented in this study is a workable design to:

- Direct and control the servomechanism by use of Analog to Digital--Digital to Analog converter interface.
- Control of the flow of information to the different display devices.
- Accomplish checking the level of correctness expected at the operator entry to the system.
- Remote processing required for the reception, processing, formatting and display of data.

The integration of the capabilities of the servomechanism, the display system in conjunction with the Analog to Digital--Digital to Analog interface allows the modelling of Shipboard Tactical systems.

The most critical technical aspects of the project was the creation of the surface and anti-air fire control algorithm, the programming of these algorithms, the simulation of the target sensor interface and finally the integration of software and hardware.

Because this study is also a learning experience, the goal was to apply knowledge acquired in different areas (e.g., microprocessor, graphics, electronic circuits, interfaces, programming, data structures and control). Therefore, attention was not paid to execution speed and high precision.

This study includes many useful procedures and modules implemented by Goncalves and Bravo [Ref. 2], which are used for graphic display, algebraic and floating point operations, conversions (ASCII to Floating Point and vice versa) and character display purposes. Some of the procedures were modified for this project.

The main contribution of this project is a closer modeling of a real system that receives information from the outside world via analog sensors, makes conversion first to binary then to floating point format, completes calculations and then converts the results back to analog form in order to control a plant (gun).

Some of the serious deficiencies in implementing microcomputer applications are:

- Low level languages for system's development.
- Primitive debugging tools to find and correct errors.
- Long compilation times for the PLM-80 programming language.

The PLM-80 programming language does not support floating point or fixed point arithmetic. This makes numerical calculations awkward to program.

Debugging tools are extremely primitive under the ISIS-II operating system. In order to use more sophisticated debugging tools available under the CP/M operating system, much extraneous and time consuming work had to be done to convert and transfer files between the operating systems.

The compilation-time for even short PLM-80 programs were of the order of several minutes. To compile a module of significant size would take fifteen to thirty minutes of execution time. Consequently program testing was done first at the procedure level, where each procedure was tested individually. Collections of procedures which comprised a meaningful module were tested as a group. Much difficulty and much time was consumed because of the lack of adequate test tools.

The analog control system was used for the simulated gun in this project. For future work, digital control appears to be a more effective way of controlling the gun.

The status of the project is: Anti-Air Fire Control and Surface Fire Control modules are ready to interface to the main body of the program. For completion; touch panel procedures, must be added to the plasma module. Procedures that calculate quadrant of contacts and draw the quadrant on the primary plasma display must be added to the ANAMOD module and to plasma modules. Finally secondary plasma display must be interfaced to system as a fire control display.

The interaction provided by the display system allowed the operator direct access to graphical data presented and to control the weapon. This capability greatly increased the operator's ability to evaluate the tactical situation and direct the weapon to the target in a real time manner.

Recommendations for follow-up work:

- Consider the replacement of the MDS 8-bit machine by a 16-bit microcomputer. This will increase the speed of execution. With this change, the speed of transmission of data should increase, which is highly desirable.

- Instead of using PLM-80 use PLM-86 or PL/I which are more powerful languages.
- The system can be modified in order to accept input data directly from the own ship's pitometer and gyrocompass in order to reduce the time required for the operator to enter the same data thus improving the system's performance.
- The system could also be modified in order to allow the use of Magnetic Bubble Memory as mass storage media, to maintain a log of all the necessary data,to be able to reconstruct events, and to start up the system again in case of a loss of power or other failure.
- Also, the capability of solving other Maneuvering Board problems such as interception, scouting and ASW attack and search patterns could be added.
- The Kalman Filter could be used as a common optimal filtering technique for tracking problems.

APPENDIX A

ALGORITHM DESCRIPTION

A. THE CLOSEST POINT OF APPROACH (CPA)

1. The Basic Relative Movement Problem

Kerns and Cooper [Ref. 13] have described a way of solving Maneuvering Board problems with the aid of a microcomputer. As described in that reference, Maneuvering Board problems are divided into two basic categories.

One is the relative plot where the CPA of contacts being tracked can be calculated. The center of the plot represents the 'reference' or 'own-ship' and any other point represents the position of a "maneuvering" ship, plotted in true bearing and range from the own ship at various times.

The other category is the vector diagram or the "triangle of courses and speeds"; this allows the operator to calculate the course and speed of any maneuvering ship (a contact) given the own ship course and speed, and relative course and speed of the contact (obtained from the relative plot).

2. Other Uses of the Maneuvering Board

All Maneuvering Board problems utilize the basic relative motion problem discussed above; in addition to determining CPA information, course, and speed of a contact, the Maneuvering Board can also be used to find the required course and speed to take station on to intercept another ship, to find true wind, or to find courses and speeds for scouting and torpedo-firing situations.

3. Maneuvering Board Problems Solutions Implemented

As described before, the system was mainly designed to provide a graphical display of the own ship and contacts being tracked; secondarily, the system provides a surface status display of information about those contacts and about own ship. Only the CPA, course, and speed of contacts are calculated automatically; the other functions of the Maneuvering Board are not duplicated.

4. Problems That Normally Occur

In a Maneuvering Board problem solution, all the data are recorded manually by the plotter; these data are provided aurally by the radar operator.

This interaction is somewhat error-prone; the positions plotted often appear scattered.

In this study air target data enter the system automatically as explained before.

5. The Least-squares Fit Approach

In order to smooth the data utilized and to obtain a straight line representing the line of relative motion for a certain number of plotted positions, the least-square fit method was chosen. This approach has been described and implemented in Goncalves and Bravo [Ref. 2].

The approach employed requires two to five contact positions. A least-squares fit is used to determine the slope and the Y axis intercept. Once these parameters are obtained the PCA, course, and speed calculations are performed in a straightforward way. According to [Ref. 14] the method is as follows:

Let N be the number of positions of a contact obtained from the radar repeater/sonar (the value of N in the systems has a bounds, 2 and 5). The equation of a straight line can be represented by:

$$y = Mx + B$$

where:

M = slope, and

B = Y intercept.

The least-squares fit method gives the solution for M and B as follows:

$$M = \frac{(s(0)t(1) - s(1)t(0))}{(s(0)s(2) - s(1)^2)}$$

$$B = \frac{(s(2)t(0) - s(1)t(1))}{(s(0)s(2) - s(1)^2)}$$

where:

$$s(k) = \sum_{i=0}^{N-1} x(i)^k , \qquad k = 0,1,2$$

$$t(k) = \sum_{i=0}^{N-1} Y(i) \cdot X(i)^k \qquad k = 0,1$$

6. CPA Algorithm

Certain combinations of data require special treatment in CPA, course, and speed calculations. These special cases are shown in Figure 24.

For Case A the contact has the same course and speed as the own ship; then the CPA can not be calculated, because the contact is permanently at CPA.

For Case B the contact has as direction of relative motion (relative course) the values of 000 or 180 degrees and thus the slope of the relative motion line will have an infinite value.

For Case C the contact has as direction of relative motion the values of 090 or 270 degrees and thus the slope of the relative motion line will have the value of 0.

There is a fourth case where the contact is on a collision course with the own ship.

In the general case, the sequence of calculations is as follows:

a. Take at least 2 marks of a contact (time, bearing, and distance).

b. Convert bearing and distance to relative values of x and y:

$$REL\$X = RNG \cdot \sin(BRG)$$

$$REL\$Y = RNG \cdot \cos(BRG)$$

c. Compute slope of smoothed relative motion line:

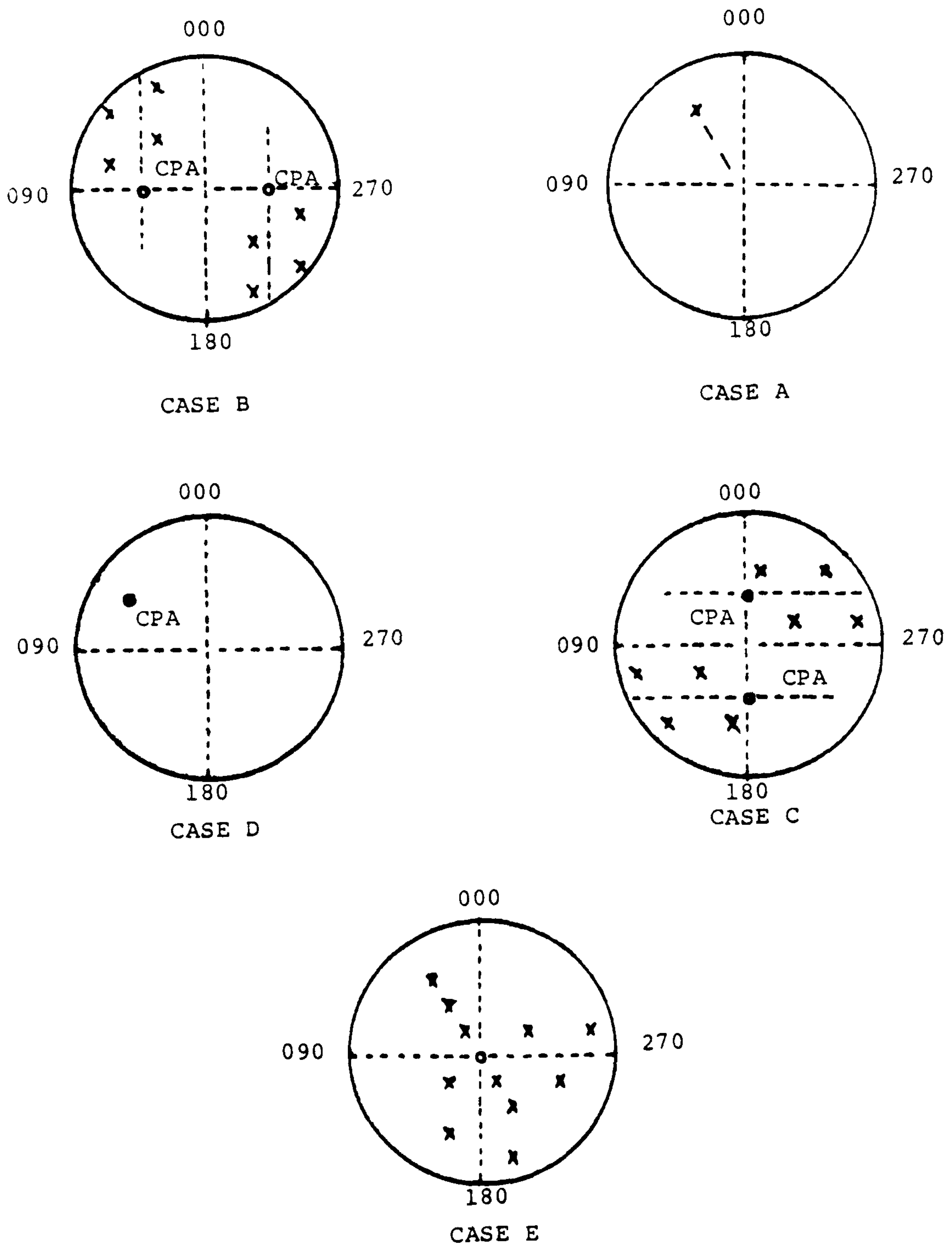


Figure 24. Special Cases in Calculation of CPA

$$M = \frac{(s(0)t(1) - s(1)t(0))}{(s(0)s(2) - s(1)^2)}$$

d. Compute Y intercept:

$$Y\$CUT = \frac{(s(2)t(0) - s(1)t(1))}{(s(0)s(2) - s(1)^2)}$$

e. Compute relative course:

$$Y1 = M \cdot REL\$X(1) + Y\$CUT$$

$$Y2 = M \cdot REL\$X(2) + Y\#CUT$$

$$REL\$CRS = \tan^{-1}\left(\frac{(REL\$X(2) - REL\$X(1))}{(Y2 - Y1)}\right)$$

where:

RELSX(2)	-	obtained from the last position used for the least-squares fit calculation
REL$X(1)	-	obtained from the first position used for the least-squares fit calculation
REL$Y(2)	-	similar to REL$X(2)
REL$Y(1)	-	similar to REL$X(1)

f. Compute relative speed:

$$DELTA\$X = REL\$X(2) - REL\$X(1)$$

$$DELTA\$Y = REL\$Y(2) - REL\$Y(1)$$

$$REL\$SPD = \frac{\sqrt{(DELTA\$X)^2 + (DELTA\$Y)^2}}{DELTA\$T}$$

where:

TIME(1) - local time at which the last position of a contact used in the least-squares fit is entered into the system.

TIME(2) - local time at which the first position of a contact used in the least-squares fit is entered into the system.

g. Compute true course and speed of a contact:

Given own ship's course (CO), own ship's speed (SO), speed of relative motion (REL$SPD), and the relative motion course (REL$CRS), then:

$$X1 = SO \cdot \sin(CO)$$

$$Y1 = SO \cdot \cos(CO)$$

$$X2 = REL\$SPD \cdot \sin(REL\$CRS)$$

$$Y2 = REL\$SPD \cdot \cos(REL\$CRS)$$

Thus, the X/Y components of the maneuvering ship's vector are X1 + X2 and Y1 + Y2 where the maneuvering ship's speed is:

$$\sqrt{(X1 + X2)^2 + (Y1 + Y2)^2}$$

and the course is:

$$\tan^{-1}((X1 + X2)/(Y1 + Y2))$$

h. Compute CPA

$$X\$CPA = \frac{(M(M \cdot REL\$X(1) - Y(1))}{(M^2 + 1)}$$

$$\text{Y\$CPA} = \frac{(\text{Y1} - \text{M}\cdot\text{X1})}{(\text{M}^2 + 1)}$$

$$\text{CPA\$TIME} = \frac{\sqrt{(\text{X\$CPA-REL\$X(1)})^2 + (\text{Y\$CPA-Y1})^2}}{\text{REL\$PD}} + \text{TIME(1)}$$

$$\text{CPA\$RANGE} = \sqrt{\text{X\$CPA}^2 + \text{Y\$CPA}^2}$$

$$\text{CPA\$BEARING} = \tan^{-1} \sqrt{\text{X\$CPA/Y\$CPA}}$$

NOTE: Y1 is the value calculated at item e, above, not the one at go.

As a final comment, the Case E shown in Figure 24 represents the situation when a contact is in collision with the own ship. It is easily seen that a contact in collision has the bearings of the various positions with approximately the same value while the range is reducing. The system was designed with a safe CPA range value (SAFE$RNG) as parameter which can be changed from 50 yards (default value) up to 1000 yards; thus, any CPA range below that parameter value will set the information about the CPA of a given contact as being in collision with the own ship.

Beyond that, the CPA algorithm checks for a contact that already passed its CPA and a message "MOVING AWAY" is issued.

B. GRAPHICS ON PLASMA DISPLAY

1. Physical Considerations

In order to provide elements for designing the algorithm for interacting with plasma display, some physical parameters for the AN/UYQ-10, Plasma Display set [Ref. 15] had to be taken into account:

a. Panel Parameters:

Active area: 8.55" × 8.55"

Addressable matrix: 512 × 512

Dot spacing: 0.0167" center-to-center, 60 per inch

Light spot size: 10 to 12 mils

b. Character size:

5 × 7 matrix: 80 × 120 mils

2. Plasma Display Unit Capabilities

Such capabilities included the capability to:

a. Set status of the Plasma Unit (busy or not)

b. Clear Plasma panel

c. Clear vectors

d. Receive X/Y coordinates from CPU

e. Set alphanumeric mode

f. Set vector mode (solid or dashed vector capability)

3. Algorithm Design

Figure 25 shows how the Plasma Panel was set in the coordinate system; the points 1, 3, 5, and 7 delimit the area where the plasma panel is located. The point 7 represents the origin of the plasma panel (ORIGIN$X, ORIGIN$Y).

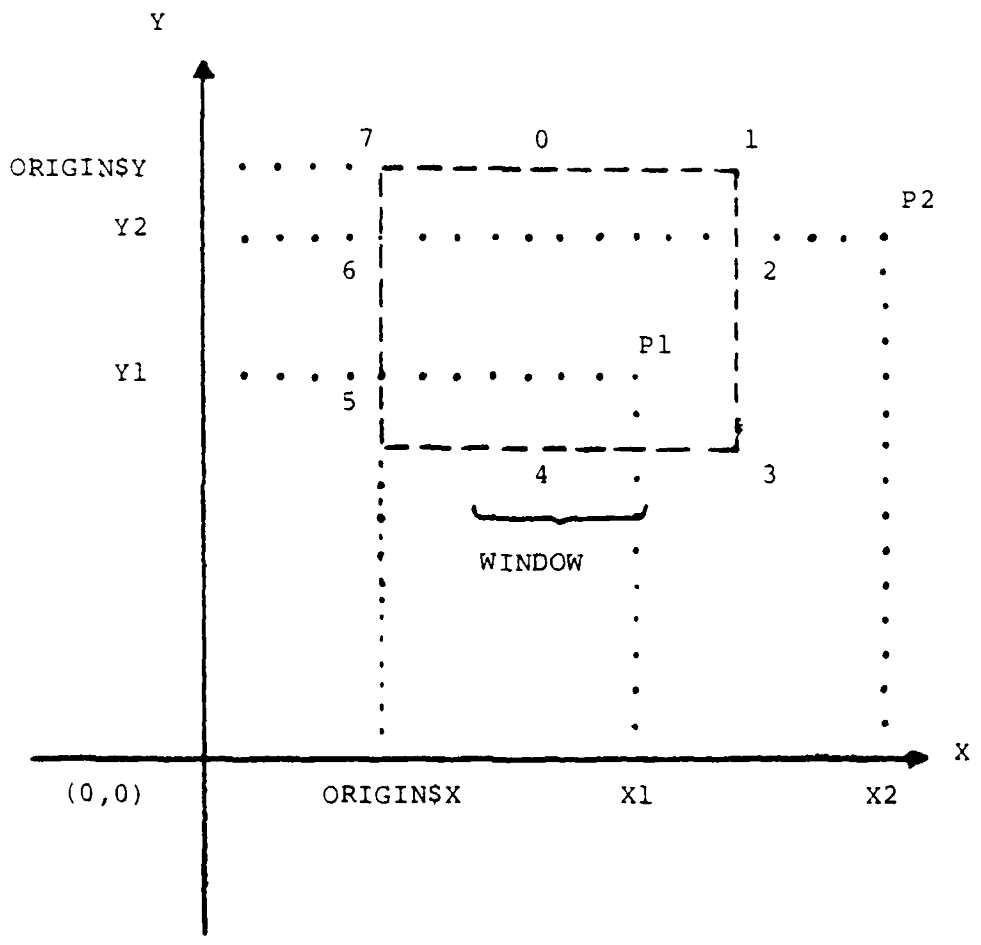


Figure 25. Windowing Schematic

In order to allow the display of all the information necessary to the Plasma Unit and generated by the system, the "PLASMA$MODULE" module was designed.

a. Windowing

The windowing process was developed as a transformation process which enables the Plasma Panel to cover a region in the coordinate grid system.

The scale set by the operator as an initial parameter controls the windowing process and it can vary from .25 miles/inch up to 25.00 miles/inch.

In Figure 25, the name WINDOW marks the size of a square representing the region covered by the Plasma Panel; this value was obtained by setting:

$$\text{WINDOW} = \text{SCALE} \times 8.55$$

b. Procedure to check if a given position falls within the limits of the defined "window".

A mechanism was implemented to check if it was possible to plot a given position (X/Y values) in the region covered by the Plasma Panel; thus, the following algorithm was developed.

In Figure 25 the point named P1 can be plotted in, but the Point P2 can not. A point can be displayed at the plasma panel when the coordinates X/Y of that point follow the rules below:

$$\text{ORIGIN\$X} + \text{WINDOW} > X > \text{ORIGIN\$X}, \text{ and}$$

ORIGIN$Y - WINDOW < Y < ORIGIN$Y

Notice that all those values are in floating point representation.

c. Normalization

Before displaying a given point in the Plasma Panel, it is necessary first to check if the position could be plotted, and secondly to normalize its value to the range specified (addressable matrix--512 × 512); the first is done as explained above, and the second is as follows:

```
(1)  DELTA$X  =  X - ORIGIN$X
     DELTA$Y  =  ORIGIN$Y - Y
(2)  Take the absolute values of
     DELTA$X and DELTA$Y:
     DELTA$X  =  ABS(DELTA$X)
     DELTA$Y  =  ABS(DELTA$Y)
(3)  TEMP$X  =  (511.0/WINDOW)·DELTA$X
     TEMP$Y  =  (511.0/WINDOW)·DELTA$Y
(4)  Truncate and convert to integer representation
     X  =  INTEGER(TEMP$X)
     Y  =  INTEGER(TEMPSY)
```

d. Plasma Reorientation

Besides setting a scale for the "window", three ways of positioning the "window" in the coordinate grid system were implemented.

By default, every time the scale has to be changed, the last position of the own ship will be set at the center of the "window"; this is accomplished by setting:

```
ORIGINS$  =  OWN$SHIP$X - HALF$WINDOW$Y, and
ORIGIN$Y  =  OWN$SHIP$Y - HALF$WINDOW
```

Notice that the own ship's last position can be set at the center of the "window" anytime the operator wants to do so.

The second method implemented was to set the last known position of any contact at the center of the "window"; this was obtained by setting:

```
ORIGIN$X  =  CONTACT$POSI$X - HALF$WINDOW, and
ORIGIN$Y  =  CON"ACT$POSI$Y - HALF$WINDOW
```

In case of no contact being maintained by the system, this method will be ignored by the system, even if requested.

The third method implemented was to set one of 8 fixed positions (refer to Fig. 25) at the center of the "window" as requested by the operator; this was obtained by setting:

```
(1)  Point 0:
     ORIGIN$X  =  ORIGIN$X, and
     ORIGIN$Y  =  ORIGIN$Y + HALF$WINDOW
(2)  Point 1:
     ORIGIN$X  =  ORIGIN$Y + HALF$WINDOW, and
     ORIGIN$Y  =  ORIGIN$Y + HALF$WINDOW
(3)  Point 2:
     ORIGIN$X  =  ORIGIN$X + HALF$WINDOW, and
     ORIGIN$Y  =  ORIGIN$Y
```

(4) Point 3:

ORIGIN$X = ORIGIN$X + HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y - HALF$WINDOW

(5) Point 4:

ORIGIN$X = ORIGIN$X, and

ORIGIN$Y = ORIGIN$Y - HALF$WINDOW

(6) Point 5:

ORIGIN$X = ORIGIN$X - HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y - HALF$WINDOW

(7) Point 6:

ORIGIN$X = ORIGIN$X - HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y

(8) Point 7:

ORIGIN$X = ORIGIN$X - HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y + HALF$WINDOW

C. TRANSCENDENTAL FUNCTIONS

Three transcendental functions were necessary in solving some problems by the system. These functions were sine and cosine of a given angle, and arc tangent of the ratio of two given values.

The main goals were the minimum amount of storage for the work area and the minimum execution time in performing the calculations; for these reasons, the Hastings approximations were chosen with slight modifications made to the algorithms suggested in [Ref. 4].

1. Cosine and Sine Functions

As described in the Appendix C, the procedure "COS$SIN" performs the cosine and sine of a given angle (in radians); the following steps were taken in the development of the algorithm:

a. Save the actual value of the angle.

b. Set angle to be between 0 and 2 PI radians.

c. Check for special cases--90, 270, and 360 degrees.

d. Normalize the angle for the interval 0 and 90 degrees, and save quadrant of the original angle.

e. Convert angle to semicircle units

$$A = ANGLE/PI$$

where PI = 3.141593

f. Perform Hastings approximation

$$A = (CLFA^2(C2+A^2(C3+A^2(C4+A^2(C5+A^2C¢)$$
$$))))A + A$$

where:

C1	=	0.5707963267949
C2	=	-0.6459640964727
C3	=	0.0796926087138
C4	=	-0.0046816668674
C5	=	0.0001602588415
C6	=	-0.0000034333379

g. Compute cosine and sine:

$$\cos(\text{ANGLE}) = 1.0 - 2.0z^2$$

$$\sin(\text{ANGLE} = \sqrt{1.0 - \cos^2(\text{ANGLE})}$$

h. Restore signs for sine and cosine according to the quadrants saved in d.

2. <u>Arc Tangent Function</u>

As described in Appendix J, the procedure "ARC$STAN" performs the arc tangent function of a given ratio (Y/X) of 2 parameter values; the following steps were taken in the development of the algorithm:

a. Save the actual values of the parameters.

b. Save sign of parameters to determine quadrant.

c. Check for valid arguments (X and Y)

(1) If $X = 0$ and $Y = 0$:

Function undefined

(2) If $X = 0$ and $Y \neq 0$:

ANGLE = 90 degrees (for $Y > 0$)

ANGLE = 270 degrees (for $Y < 0$)

d. Form Z to perform the Hastings approximation

$$Z = \frac{Y - X}{Y + X}$$

e. Perform the Hasting approximation

$$\text{ANGLE} = (C1+Z^2(C2+Z^2(C3+Z^2(C4+Z^2(C5+Z^2(C6+Z^2(C7+Z^2C8)))))))Z + PI/4$$

where:

C1	=	0.9999993329
C2	=	-0.3332985605
C3	=	0.1994653599
C4	=	-0.1390853351
C5	=	0.0964200441
C6	=	-0.0559098861
C7	=	0.0218612288
C8	=	-0.0040540580
PI	=	3.141593

f. Restore angle to proper quadrant

D. POSITIONAL DATA CONVERSION

In the design of the system all the positions can be referred either as latitude and longitude, or as X/Y coordinates. For this reason, some algorithms were developed in order to obtain one or another kind of positional data.

1. Convert LAT and LONG to X/Y Coordinates

The whole system was based on a Coordinate Grid System whose origin values were given in terms of latitude and longitude, and any position in it had an X/Y coordinate defined in relation to the origin; thus, given the values of latitude and longitude of a certain position, it might be converted to that Coordinate Grid System units; i.e., to convert to X/Y coordinates. This was obtained by doing:

a. Compute mean latitude:

```
MEAN$LAT = (SYSTEM$LAT + LAT)/2.0
```

b. Compute X/Y coordinates:

X = (LONG-SYSTEM$LONG)·cos(MEAN$LAT)

Y = (LAT - SYSTEM$LAT)

2. <u>Convert a Given Position in Terms of Bearing and Range from Own Ship to X/Y Coordinates</u>

In order to determine the X/Y coordinates of a position when it is given in terms of bearing and range from the own ship, the following steps were done:

a. Save value of bearing:

ANGLE = BEARING

b. Compute DELTA$X and DELTA$Y:

DELTA$X = RANGE·sin(ANGLE)

DELTA$Y = RANGE·cos(ANGLE)

c. Compute X and Y:

X = OWN$SHIPSX + DELTA$X

Y = OWN$SHIPSY + DELTA$Y

3. <u>Convert X/Y Coordinates of a Given Position into Latitude and Longitude</u>

a. Compute latitude:

LAT = Y + SYSTEM$LAT

b. Compute mean latitude

MEAN$LAT = (SYSTEM$LMT + LAT)/2.0

c. Compute longitude:

LONG = X/cos(MEAN$LSAT + SYSTEM$LONG)

APPENDIX B

INTERRUPT STRUCTURES

A. BACKGROUND INFORMATION

A very large percentage of microprocessor systems are employed in control applications, i.e., situations in which the CPU controls a process in the "real world" by analyzing information concerning the behavior of the process.

Information is a measure of "surprise", and in many systems this equates as much to "when" as to "how much". To find out "when" an event occurs, it is possible to read and test status bits that are set by the events occurrence. For a number of systems this may require almost continuous sampling while only a relatively few samples return much information. The machine can do no useful work while sampling and, thus, is inefficiently utilized. By allowing the events of interest to gain the machine's "attention", the efficiency can be vastly improved. Thus, interrupt structures allow "event driven" systems for which the concept of temporal continuity has little relevance.

An interrupt signifies either the occurrence of an interval operating system event such as the completion of a process or the status of the Real Time Systems clock, or the readiness of a peripheral device such as a Teletype or a CRT to communicate data to or from the computer. The ability to respond to "external" events relative to the execution of the current process allows processors to be shared by one or more processes if a means can be found to handle "simultaneous" events, i.e., those events occurring within one basic system cycle normally the current instruction cycle. The usual means of handling simultaneous "interrupt" requests is by embedding the "concurrent" processes within a priority structure. The priority structure can be implemented in hardware and/or software.

The functioning of an interruptible computer program can be viewed as similar to that of the job of a secretary. The secretary has a scheme of priorities about her work, higher priority items being serviced on a more immediate schedule than lower priority items. Imagine the situation of a letter being type when the phone rings. The activity in progress, the letter, is suspended while the immediate demand of the telephone is serviced. When the phone call has been completed, lower priority typing activity is resumed. Some items have ultimate priority in this scheme, a fire alarm for example. No sane person bothers to answer the phone or even less to continue typing a letter when the building is burning down. Another important feature of the secretary's work environment is that nothing is of such priority that it must be done instantaneously. If the phone rings in the middle of a typed word or while the typist is taking a sip of coffee, the work is finished or the cup set down on the desk before picking up the phone. This illustrates an important feature of interruptible environments. However, the interruption is serviced, it must not cause a disruption of the former activity in such a way that it cannot be successfully resumed.

When a program is interrupted, the presumption is that the cause of the interruption is of some immediate priority, but not of such priority that the lower priority task being interrupted needs to be disrupted. One thing therefore needs to be understood at the outset about interrupts. The instruction in progress when the interrupt request arrives is always finished before the interrupt is honored. When the interrupt request arrives, the instruction in progress is always finished before the interrupt is honored. When the lower priority activity was resumed after servicing the interrupt, the program would have no means of rectifying the damage done by the half-completed instruction.

The most important single thing that the programmer must remember about interrupted programs is that the status of the interrupted program must be preserved. The program which services the high priority interrupting activity will use the same registers and flags to accomplish its task as the interrupted program uses. These registers and flags must be restored to their condition at the time of the interrupt, before returning control to the former activity, or the former activity will be disrupted. Failing to observe this caution is the most common single error in programming for interrupt driven systems. The saving and restoring of the status of the interrupted program is of crucial importance.

A very common type of interrupt is that caused by some event external to the program which does not require that data be transmitted. The event itself constitutes the required information. In this category are such applications as traffic counters. The passage of a car through the sensor of an expressway ramp does not require the transmission of data for that car to be counted in the flow. In this case the interrupt itself constitutes notice to the system that a car has passed the sensor and that the counter is to be incremented. A similar situation is encountered in devices that register angular position through the counting of passing gear teeth. Exactly which tooth has passed is not of any interest, only the fact that a tooth has passedis of consequence.

Perhaps the most common of these event counting situations occurs with the computer option called a real time clock. The real time clock is not a clock in the common sense of that word. It does not keep time at all, but simply generates a series of pulses at uniform intervals, these pulses causing interruption of the operating program at these uniform intervals. The program which counts the interrupts can use this counting to keep a programmed time of day clock.

The elementary process that is crucial to an interrupt structure is the CALL/RETURN transfor-of-control process. The subroutine is a sequence of code that is executed upon the invocation of its name and that returns control to the calling sequence upon completing its execution. An interrupt process can be thought of as an unexpected or surprise subroutine call. In a program, the invocation is accomplished by inserting a call instruction at a known position in the instruction sequence. During interrupt processes, the invocation will occur at unknown positions in the control sequence. Thus, provision must be made for saving the return address in a known location for later retrieval. Mathematics can be described as a "replacement" process in which the replacements are made under control of the mathematician. Interrupt systems are those in which the replacement of a given control sequence by another can be made upon request from any external system. The complete control sequence is composed of a set of elementary sequences, or control strings, that can be edited by real-world systems to adapt to local conditions.

Varieties of interrupt structures are designed with one goal in mind: to share one CPU efficiently between several "concurrent" processes. This can be accomplished via this procedure:

- save state of current process;
- identify device requesting service;
- transfer control of CPU to this device;
- upon completion of service, restore state; and
- transfer control of CPU back to interrupted process.

Although minor variations exist in implementation of these steps, they are always executed. This procedure is shown diagrammatically in Figure 26.

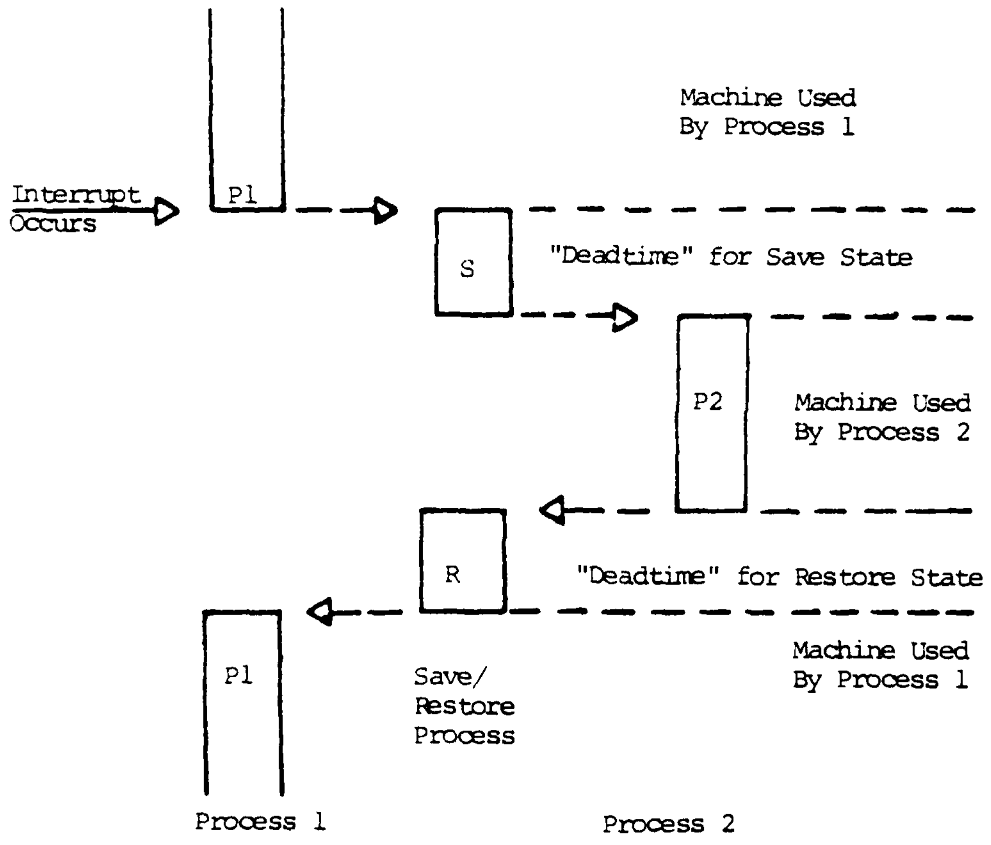


Figure 26. Interrupt Process Scheme

B. PRIORITY INTERRUPTS

There are two basic implementation strategies for priority interrupts: Polled priority interrupts and Vectored priority interrupts.

1. Polled Priority Interrupts

Polled priority interrupt methods trap (acknowledge and jump) all interrupts requests to a common location, and a routine that POLLs status bits determines the source of the interrupt requests. If the interrupting devices can be arranged in a hierarchical order, then the highest priority device will be polled first, the next highest will be polled second and so on. Thus, if two devices request service at once, the higher priority will be encountered first in the poll and it will receive service first. It should be noted that this method does not involve clearing the interrupt request.

2. Vectored Priority Interrupts

An interrupt system in which the hardware supplies a separate address for each interrupting device is called a VECTORed interrupt structure as opposed to the POLLed structure in which all devices trap to the same address, and device identification and conflict resolution are accomplished in software.

The use of hardware to encode these instructions for separate devices will speed up interrupt servicing by eliminating the need to POLL the devices. In addition, standard harware is available for conflict resolution.

If two devices simultaneously request service, a priority encoder will pass only the higher priority request. This feature by which higher priority devices can interrupt lower priority devices, but lower priority devices cannot interrupt higher, is common to most interrupt structures.

Each device can have a unique address associated with its service routine, and the hardware just described automatically provides a 1-byte call instruction that causes transfer of control to this address. VECTORed interrupt system provide the fastest possible interrupt servicing, because no time is wasted polling status bits.

Since the highest priority requests override all others there must be a means of individually removing each request as it is serviced, so that lower priority requests can be seen. The lower requests must, therefore, remain active until their time comes.

C. 8080 MICROPROCESSOR INTERRUPT METHOD

The 8080 microprocessor interrupt method is described below to illustrate a VECTORED PRIORITY INTERRUPT STRUCTURE implemented in a microcomputer.

An interrupt can occur when all of the following three conditions are met. If any of them is not met the interrupt cannot occur. They are:

1. The 8080's interrupt system has been enabled by the use of EI (Enable Interrupts) instruction. The form enable has specific application to the interrupt system as a whole and

is not applicable to any specific device or peripheral. The interrupt system is enabled by the EI instruction and disabled by the DI (Disable Interrupts) instruction. In the disabled state no interrupts from any source can occur.

2. The specific device or peripheral interface has been conditioned by the program in such a way as to be able to generate interrupting pulses. This conditioning is known as arming. An interface which has been so conditioned is said to be armed. If the interface has not been conditioned so as to be able to generate interrupting pulses it is said to be disarmed.

3. The device or peripheral ready flag is set by the event which is to cause the interrupt.

Again, in the absence of any of the above conditions there is not and cannot be an interrupt. When the above conditions are all met, the following sequence of events takes place:

1. After finishing the execution of the instruction in progress at the time all of the conditions for interrupt were met, a special instruction is forced into the instruction register (I). This instruction is provided by the hardware itself and is not resident in computer memory. The program counter is not changed by this. It still points to the next program instruction to be executed, i.e., the instruction following the one in progress at the time the interrupt conditions are met.

2. The special instruction forced into the instruction register is executed. This special instruction is a kind of

CALL whose target address is explicitly given in the instruction. The special call, known as a restart, pushes the program counter onto the stack just as a normal CALL does. Instruction execution then starts at the address which was implicit in the interrupt instruction.

APPENDIX C

DESCRIPTION OF GRAPHIC DISPLAY TECHNOLOGY

A. GENERAL INFORMATION

Most graphic display systems use refresh or storage technology. Three main types of refresh technologies exist: stroke writing, roster scanning and scan converting. Stroke writing display systems position an electron beam on the tube face much as one would draw on paper with a pencil. In roster scanning systems, the beam sequentially traces the entire face of the tube. When the beam arrives at a point that belongs to the picture under construction, a video signal brightens the beam to illuminate the screen. Hybrid scan converters use a storage tube to store the image and then scan the storage tube information onto a roster scanning monitor to display the image. Since the persistence of the phosphor in the tube is low, CRT's using one of these technologies require periodic image refreshing to prevent annoying screen flicker. These CRT's refresh the image at least forty times each second.

Two storage technologies exist: the storage tube and the plasma panel. With the storage tube, the CRT receives its image in the same way as a stroke writing system. However, the storage tube stores the image on a grid, eliminating periodic refresh. Unlike other graphic display systems, plasma panels do not use CRT's. The display consists of a series of bright data that can be formatted into alphanumerics and graphics. Plasma panels do not require refresh and, once a particular point on the display is "turned on", it continues to glow until "turned off".

B. PLASMA PANEL PHYSICAL DESCRIPTION

1. Panel Parameters

a. Actual area; 8.55" × 8.55"

b. Panel Glass size: 12.25" × 12.25"

c. Light Spot size: 10 to 12 mils

d. Addressable Matrix: 512 × 512

e. Dot Spacing: 0.0167" center-to-center 60 per inch

f. Brightness: 50 foot Lamberts

g. Contract ratio: 25 to 1 (nominal)

h. Color: Neon-range (585.2 monometers)

2. Character Size (Other Sizes Dependent Upon Driving Logic and/or Software)

a. 5 × 7 matrix: 80 × 120 miles

b. 7 × 9 matrix: 120 × 150 mils

3. Electric Descriptions

The Plasmascope primary power requirements are:

a. 115 V ac

b. 47 to 440 Hz

c. 300 watts maximum

d. single phase

4. Performance

a. Data Rates: Addressable to the individual dot data at 50 KHz;

b. Parallel Mode: 330 msecs to address the entire screen. The parallel mode of operation allows the simultaneous addressing of 16 points in the Y axis, Y_0, Y_8 and Y_4 through Y_7 (Y_0,Y_8) of the Y address are used to select one of 32 sectors, each of which comprises 16 consecutive horizontal electrodes. The X address selects one column of 16 points in the addressed sector. The parallel address inputs are then used to address any number of the 16 points in the selected sector column.

5. Data Code

ASCII character set for alphanumeric operation (7 bit).

6. Character Matrix

5×7 or 7×9 dot matrices

7. Reliability

The mean-time-between failure (MTBF) for the plasmascope is over 6,000 hours with JAN-TX parts at 25°C.

8. Maintainability

The mean-time-to-repair (MTTR) for the plasmascope is 2 hours, board level maintenance or on-board replacement.

9. ELECTROMAGNETIC INTERFERENCE/TEMPEST

The plasmascope has been tested to MILSTD-461 for EMI suppression. Several aspects of the model 2500 design are more critical for Tempest than for EMC. These include the display panel, the keyboard, and the power line. The display panel contains grid wires, approximately 8 inches long, that contain signals which correlate with the information being displayed. A metallic film on a face plate is provided for the display panel to minimize radiation. This precaution has been sufficient to permit other displays of similar design to meet the Tempest requirements.

The keyboard is somewhat exposed to radiation as discussed above for EMC. Because the exciting signal levels are small (HALL Effect voltages) and because the radiating elements are electrically very short at the processing signal frequencies, the keyboard is expected to satisfy Tempest requirements.

The power level conductor requirements of Tempest are met by a combination of filtering and consideration of Tempest requirements in the design of power supply.

The display electronics operates at 50 kHz, a frequency at which the EMI filters provide attenuation. The data lines between the Model 2500 and any data source can be designed for Tempest by providing adequate cable shielding to contral EM radiation.

C. PLASMASCOPE TOUCH PANEL DESCRIPTION

The device electronics can be divided into three sections: the scanning system, consisting of the oscillator-counter; the light sources and the detectors; and control logic.

The scanning system eliminates the optical collimation problem usually associated with a light-grid touch panel. The system only activates one light source/detector at a time. Time is the means of separating the light beams rather than a complex optical collimation system. This scanning is controlled by a free running oscillator driving a 4-bit counter. The output of the counter is used to sequentially select the light source/detector pairs, and to provide the address.

The light sources are infrared light emitting diodes (LED) chosen for their high output power, cast, and package design. Since these devices are diodes, a diode matrix drive scheme is used to reduce complexity. The output of the counter activates the appropriate transistors and causes two of the diodes to turn-on; one diode is in the x array, the other in the y array. In this way each diode pair (one x, one y) is sequentially pulsed and the display is scanned.

The detectors are silicon phototransistors similar to LED in package design. The detectors are located across from the LED's in a plastic frame which fits around the display. Four detectors, spaced evenly along the side, share a common amplifier. The output of the four amplifiers are time multiplexed so that the proper amplifier is actuated at the correct time. Only four amplifiers are needed because of the natural optical collimation associated with the plastic frame. For example, detectors #0, 4, 8 and 12 are activated when LED #0 is pulsed. Light from #0 LED is received by the #0 detector. The other detectors sharing the amplifier with #0 receive very little light from #0 LED.

The amplifier feeds the signal from the detector to a voltage comparator which can have one of two voltage thresholds. The purpose is to introduce scanning hysteresis eliminates false inputs due to room light partially broken beams. It works by setting up two conditions; one for initially detecting if the light beam is broken, and one for subsequentially deciding when it is not broken. When initially scanning, the detect level is set low so that a beam to be detected broken must be completely absent. Upon the detection of both an x,y broken beam, the threshold voltage is raised to a higher level. Now a beam to be detected again must be larger than this higher threshold. In this way, marginal signals are ignored and only a beam, either absent or present is detected.

The basic operation of the total system is to sequentially activate pairs of source detectors on both the x and y axes by means of the scanning logic. When a broken beam is detected, the address (or position) is stored in the appropriate storage register. When both an x and y beam are broken, the information is sent to the computer. Scanning continues and with each scan, any new broken positions are compared with the old position stored in registers. If the positions agree (i.e., the obstacle has not moved) scanning continues; if they disagree (the obstacle has been removed or shifted to a new location), the system resets. Touch inputs for the touch panel are limited to 10/sec by a short delay before reactivating.

APPENDIX D

ANALOG INPUT/OUTPUT SYSTEM DETAILED DESCRIPTION

A. THEORY OF OPERATION

When programming with these peripherals, they are treated as memory locations. Any memory reference instruction can be used. Both the A/D converter output and the D/A converter input are 8-bit words so one memory location is needed for each channel. Because the address block occupied by each peripheral is user selectable, it can be placed anywhere in memory.

Because these units are treated as memory, a minimum of instructions are needed to read an input channel or to set the input of a D/A converter. For instance, LHLD (load) instruction followed by the proper address can be used to read data from two successive analog input channels. It will automatically select the desired channel, initiate conversion and when conversion is complete, transfer the A/D converter output for the first channel to the 8080's L register and the second channel to the H register. Likewise a single LDA instruction can be used to read one analog input channel.

All of these systems are jumpered at the factory with the first channel at address F700H. Each subsequent channel is one memory location past the start of the last channel so that the second channel is at location F701H.

B. OPERATING INSTRUCTIONS

1. Installations

These units are calibrated and ready for use. Installation requires only plugging the card into any empty slot in the computer and wiring the analog connector.

2. Programming

Programming of this analog I/O board is easily accomplished since all channels are treated as memory locations. Any memory reference instruction can be used. A single STA instruction may be used to load the accumulator contents to one of the D/A converters. Likewise a single LDA instruction can be used to read an analog input channel.

Single instructions can also be used to set the inputs of both D/A converters and read two adjacent analog input channels. An SHLD instruction referenced to DAC 1 will load the contents of the L register into DAC1 and the contents of the H register into DAC2. An LHLD instruction will read the channel addressed and the next higher channel. The channel addressed will be transferred to the L register and the next higher channel to the H register. Of course, any MOV instruction may also be used if direct addressing is not desired.

The normal operation of this board halts the CPU during the conversion time of the analog input system. This is because the software in this mode is simpler than in any other (i.e., only one instruction required). If the halt feature is not desirable it may be disabled.

For operation without halting the CPU, the conversion should be started by using a single channel memory reference instruction (LDA or MOV). Then the CPU should execute a routine which will take longer than the conversion time (44 to 84 microseconds). When the CPU now uses an LDA or MOV referenced to the same memory location, the converted data will be transferred to the CPU.

The voltage data for these boards is represented by an 8-bit two's complement binary number. With ±5 range, each bit has a value of 39.1 mV, with the polarity of the voltage indicated by the sign of the binary number.

Each board is set at the factory for a block of addresses beginning at F700H. Any analog data channel requires one memory location. Thus the first analog channel is located at F700 while the second analog channel is located at F701.

3. Address Modification

The base address of a board can be set to any value by properly jumpering its address selector. The most significant 8 bits of the address (ADR/8-F) are jumpered to read F7 by plated through connections on all boards. These addresses can be changed by first drilling out the hole that makes the connection and then soldering a wire jumper between the bit and logical zero or one.

4. Analog OUTPUT/INPUT RANGE SELECTION

Normally DAC is jumpered for ±10 Volt operation (two's [illegible]ment coding). However it is possible to alter these jumpers for other output voltages and coding (±5 V, ±2.5 V, 0 to 10 V, 0 to 5 V).

Two's complement coding is typically used for bipolar ranges and straight binary for unipolar ranges, but either coding can be used for any range. Analog full scale Range Values as shown Table D-3.

The analog input system can be set for any range between ±5 V and ±2.5 mV. It is set for ±5 V (two's complement coding) from the factory. There are two gain determining

Table D.1

ANALOG FULL SCALE RANGE VALUES

BIPOLAR--TWO'S COMPLEMENT

Digital Input Output	±10 V	±5 V	±2.5 V
0111 1111 (7FH)	+ 9.922 V	+4.961 V	+2.480 V
1000 0000 (80H)	-10.00 V	-5.000 V	-2.500 V

UNIPOLAR--STRAIGHT BINARY

Digital	Input/Output	0 to +10 V
11111111 (FFH)	9.961 V	4.980 V
00000000 (OOH)	0.000 V	0.000 V

elements in this system: the A/D converter and the instrumentation amplifier (IA). The A/D converter is set for a ±10 V range and the IA for a gain of 2 at the factory. The A/D converter can be set for other ranges simply by changing jumpers. System block diagram is shown in Figure 27.

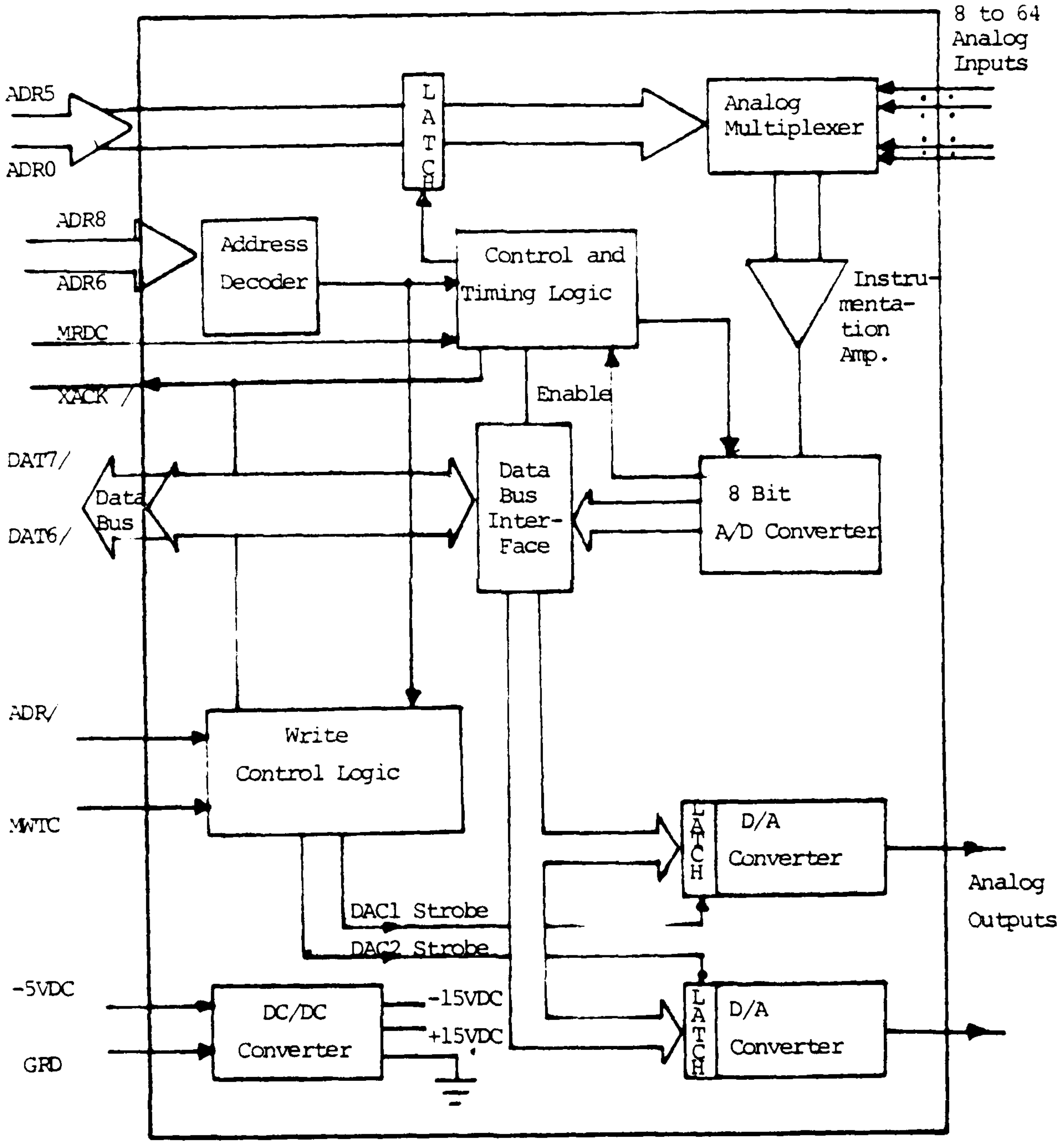


Figure 27. Analog Input/Output System Block Diagram

C. SPECIFICATIONS

1. Analog Input

Number of analog inputs

- 8 differential
- 16 single-ended
- 32 differential or 64 single-ended

Input Voltage range: ±10 mV to ±5 V

ADC gain ranges: ±10V.0 to 10V.0 to 5 V

(strap selectable): ±5 V ± 2.5 V

Amplifier gain range: 1 to 1000

(Resistor programmable): G = 100 kΩ R_{ext}

Amplifier gain equation: (Resistor programmable)

±15 V

Input overvoltage protection: 100 megohms

2. Analog Input Transfer Characteristics

Resolution: 8 bit binary

Throughput accuracy:	±5 V range (max)	±0.4
	±10 mV range	±0.5
Temperature coefficient of accuracy		
	±5 V range (max)	±0.02
	±10 mV range	±0.07
Conversion time	±5 V range	44 microseconds
	±10 mV range	84 μsec

3. Analog Output

Number of analog outputs: 2

Output voltage range: ±10 V, 0 to 10 V, ±5 V, 0 to 5 V, ±2.5 V at 5 mA (strap selectable)

Output impedance: 1Ω

Output settling time (max) <5 microseconds

4. <u>Analog Output Transfer Characteristics</u>

Resolution: 8 bits binary

Throughput accuracy (max) ±0.4% FSR

5. <u>Digital Input/Output</u>

All signals are compatible with Microcomputer bus.

Output Coding: Bipolar two's complement;
Unipolar, straight binary

An analog input channel is selected by: ADRO
through ADR5

An analog output channel is selected by: ADRO

The input/output data bits are read through:
DAT0 through DAT7.

APPENDIX E

FLOATING-POINT HARDWARE BOARD DESCRIPTION

A. GENERAL INFORMATION

The floating-point package developed for this system is based on the SBC-310 High-Speed Mathematics Unit from Intel Corporation. As described by Reference 16, the SBC 310 Unit is a member of a complete line of the Intel SBC 80 System expansion modules. In performing high-speed mathematical functions, the Math unit acts as an intelligent processor slaved to one or more SBC 80 Computer Masters. The Mathematics Unit performs its repertoire of 14 arithmetic functions an order of magnitude faster than is possible with software routines.

B. DESCRIPTION OF THE MATH UNIT

The Math Unit is a microprogrammed processor on a single board and is designed to be plugged into a standard SBC 604/614 Modular Backplane and cascaded to interface directly with a SBC 80 single board computer or to be used with an Intel Intellec Microcomputer Development System (MDS).

The Math Unit includes the following standard Intel Series 300 Schottky bipolar components: 3001 Microprogram Control Unit (MCU), 3002 Control Processing Element (CPE), 3003 Look Ahead Carry generator (LCG), and 3604 Electrically Programmable Read only Memory (PROM). Also included are pipe line register and lens interface logic. The pipeline register permits the overlapping of microinstruction fetch/execute cycles and the bus interface logic provides compatibility with the Intel Multibus.

Standard Operations include floating point add, subtract, multiply, divide square and square root; fixed point integer multiply, divide, and extended divide; conversion between fixed and floating point representations; and test, compare, and argument exchange operations.

The Math Unit implements unbiased rounding for maximum accuracy. Unbiased rounding is the same as ordinary unless the result is exactly midway between two floating point numbers; in this case ordinary rounding always increases the result, whereas unbiased rounding rounds the result to the nearest even number. When a calculation is performed that results in either an exponent underflow or overflow, the Math Unit provides exponent warparound to prevent loss of information.

Operation Codes for invoking the arithmetic functions are passed to the Math Unit via I/O Write Commands, which are also used to initialize the unit with a memory base address. I/O read commands are used to determine the Math Unit status. Arguments are passed to the Math Unit via Memory Write, Commands and the results are obtained via Memory Read Commands.

The Math Unit which can be operated either in the Interrupt of POLLed mode, generates a busy during processing operations and generates either complete signal or an Error signal after the computation is complete. The information to the host computer which these three signals convey is explained in [Ref. 16].

The memory base address and I/O base address are user selectable. The 16-bit memory address is completely under software control and is assigned by the host processor through a sequence of I/O Write Commands, addressed to the Math Unit. The 3 bit I/O base address is selected by a dual inline package (DIP) switch on the board.

All Math Unit operations including arithmetic calculations, data flow between functional elements on the board bus interface, and associated logical tasks, are resident microprogram permanently stored in a set of eight Intel 3604 Erasable Programmable Read Only Memory (EPROM) chips. This memory provides 1.024 micro-instructions of 32 bits each.

Installation Consideration, I/O Base Address Switches, Programming Information Math Unit Functions, Argument and Result Data formats, Status and Flags were explained in complete detail in [Ref. 16].

APPENDIX F

OPERATOR'S MANUAL FOR THE MULTIPURPOSE CONTACT PLOTTER AND DUAL PURPOSE GUNFIRE CONTROL SYSTEM AT THE NAVAL POSTGRADUATE SCHOOL

This manual describes the operation of the multipurpose Contact Plotter and Model of Gunfire Control System at the Naval Postgraduate School. This manual assumes familiarization with CIC, Gunfire control procedures. The specifics about the installation of the equipment required were presented in Chapters IV, V and VI, also in Appendices C and D. The algorithms used were described in Chapter III and Appendix A. All the software required is contained in two diskettes labeled PLASMA MULTIPURPOSE GEOGRAPHIC PLOTTER AND GFCS PACKAGE: SYSTEM. APL.

I. SYSTEM START-UP PROCEDURE

Caution: Never turn on or off the diskette drive with a diskette inserted!!!

1. Set the air contact starting values on the radar simulator (Range, Bearing and Elevation).
2. Set the aircraft speed and steering values on the aircraft simulator (Vx, Vy and Vz speed components).
3. Set the Closed Loop Control System gain values as follows:

   Position feedback grain: 80%

   Speed feedback gain: 45%

   Forward path gain (K_3): 92%

4. TURN ON THE MODULAR SERVO SYSTEM: Use the power switch located at the power supply unit.
5. TURN ON MDS SYSTEM: Use the key located at the upper left corner of the front panel and turn it clockwise. The power indicator should light.
6. TURN ON DISKETTE DRIVE: Use power switch located at the front panel. The ON indicator should light.
7. TURN ON DATAMEDIA TERMINAL (CRT). Use switch located on right side. The cursor should appear at the screen after a few seconds. Ensure that the lights CD, CTS, ROLL and FULL DUPLEX are on.
8. TURN ON POWER SUPPLY TO PLASMA UNIT. This external power supply should be set at +5 Volts D.C. A red indicator should light.
9. TURN ON AN/UYO-10 PLASMA DISPLAY UNIT. Use POWER switch located at front of unit. The indicator located at the upper left corner should light.
10. Place diskette labeled PLASMA MULTI PURPOSE GEOGRAPHIC PLOTTER AND GFCS PACKAGE: SYSTEM. APL in drive 0, with the read/write access slot first. Close door of the drive after diskette insertion.
11. TURN ON SBC POWER SUPPLY UNIT. Use POWER switch located at front of unit.
12. Bootstrap the ISIS-II Operating System:
    a. Press top of Intellec BOOT switch.
    b. Press top of RESET switch.

c. Observe that INTERRUPT 2 indicator goes on before proceeding.

d. Press space bar of DATAMEDIA Video Terminal keyboard.

e. Observe that INTERRUPT 2 indicator goes off before proceeding.

f. Press bottom of BOOT switch.

g. Observe that the following message appears at the DATAMEDIA Video terminal screen:

```
ISIS-II, V2.2
-
```

13. Type the following command

    DPGFCS <CR>

14. After a few seconds, the DATAMEDIA Video Terminal screen should be cleared and then filled with working format; also, the message "ON-LINE" should appear at the AN/UYO-10 Plasma Display screen.

15. Follow the instructions for SYSTEM INITIALIZATION as prompted and according to the format explained in the following pages.

16. Notice that the TIME value entered during SYSTEM INITIALIZATION should be that one desired as starting time (the time at which the 60 key is depressed, after the SYSTEM INITIALIZATION mode is completed).

17. During operation, the INTERRUPT 1 indicator should light (after the GO key has been depressed to start the system).

II. SHUTDOWN PROCEDURE

1. Press the INTERRUPT 0 switch. The associated indicator should light.
2. Eject the diskettes in drives 0 and 1.
3. Turn off the equipment in the following order:
   a. AN/UYO-10 Plasma Display Unit
   b. AN/UYO-10 Plasma Display Unit Power supply.
   c. DATAMEDIA Video Terminal.
   d. Diskette drive.
   e. SBC power supply unit.
   f. Intellec MDS system.
   g. The Modular Source System

III. FORMATS AND COMMAND DESCRIPTION

The following pages describe the data elements, input commands, and display commands required for the operation of the MULTIPURPOSE CONTACT PLOTTER AND GFCS SYSTEM.

Input and display keys arrangement are shown in Figure 28.

A. DATA ELEMENTS

1. Time Zone Number

Parameter defining the time zone number being used to determine the local time.

FORMAT: SNN

where: S--Sign (+ or 0)

NN--Two digit number

RANGE: $00 \leq NN \leq 12$

2. Time

Parameter defining a time value. Consists of hours, minutes and seconds.

Once the time is set, the system will maintain the current timeand update the time value displayed at the Video Terminal every second.

FORMAT: HH:MM:SS

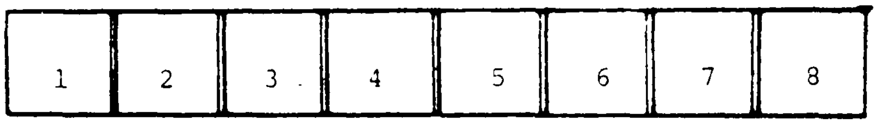
where:

HH--hours

MM--minutes

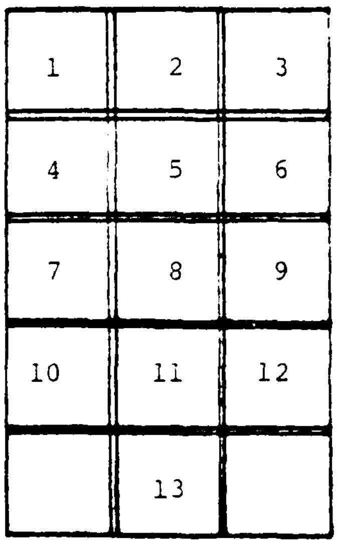
SS--seconds

DISPLAY KEYS



1. Origin	2. Scale
3. Own Ship	4. Contact Information
5. Contacts in Systems	6. Safe CPA Range
7. Wind	8. Display Update Time

INPUT KEYS



1. Create a Contact	2. Modify Own Ship
3. Modify Coordinate Grid Origin	4. Remove a Contact
	6. Wind
5. Contact to Display	8. Time
7. Redesignate a Contact	10. Update a Contact
9. Scale	12. Plasma Reorientation
11. Safe CPA Range	13. GO Key

Figure 28. Input and Display Keys Arrangement

AD-A102 693 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
MICROCOMPUTER BASED SHIP-BOARD GUN CONTROL SYSTEM.(U)
MAR 81 A ERDOGAN
UNCLASSIFIED NL

3 OF 7
AD
A102693

RANGE: $00 \leq HH \leq 23$

$00 \leq MM \leq 59$

$00 \leq SS \leq 59$

3. Time Between Updates

Parameter that defines the interval of time used by the system to update the geographical position of the own ship.

Initially, the Time Between updates is set automatically by the system to 180 seconds.

FORMAT: SSS

where:

SSS--seconds

RANGE: $015 \leq SSS \leq 250$

4. Course

Parameter that determines the general direction at which the own ship or any contact is steering.

FORMAT: DDD.D

where:

DDD.D--degrees and tenths of degrees.

RANGE: $000.0 \leq DDD.D \leq 359.9$

5. Speed

Parameter that determines the velocity at which the own ship or any contact is moving.

FORMAT: KKK.K

where:

KKK.K--knots and tenths of knots.

RANGE: $000.0 \leq KKK.K \leq 999.9$

Needs to be entered from keyboard for the own ship, from target simulator for the air target. The system will calculate its value in the case of a contact.

6. Bearing

Parameter that determines the true bearing of a contact from the own ship, at a given time.

FORMAT: BBB.B

where:

BBB.B--degrees and tenths of degrees.

RANGE: $000.0 \leq BBB.B \leq 359.9$

7. Range

Parameter that determines the distance between the own ship and any contact, at a given time. Can be given in yards or in nautical miles.

FORMAT: MMM.M or YYYYYY

where:

MMM.M--miles and tenths of miles

YYYYYY--yards

RANGE: $000.0 \leq MMM.M \leq 100.0$

$000000 \leq YYYYYY \leq 199999$

8. Latitude

Parameter that defines the geographical position of the own ship or any contact. Consists of a sign, degrees, minutes and tenths of minutes.

FORMAT: DD:MM.M S

where:

DD--degrees

MM--minutes and tenths of minutes

S--sign (North (N) or South (S))

RANGE: $00 \le DD \le 89$

$00.0 \le MMM.M \le 59.9$

9. <u>Longitude</u>

Parameter that defines the geographical position of the own ship or any contact. Consists of a sign, degrees, minutes and tenths of minutes.

FORMAT: DDD.MM.M S

where:

DDD--degrees

MM.M--minutes and tenths of minutes

S--sign (East (E) or West (W))

RANGE: $00 \le DDD \le 179$

$00.0 \le MM.M \le 59.9$

10. <u>X</u>

Parameter that defines the position of the own ship or any contact in the Coordinate Grid System being used. Its value is given in miles.

FORMAT: SNNNNNNNNNN.NN

where:

S--sign (+ or -)

NN--miles and hundreths of miles.

RANGE: $0.0 \le NNNNNNNNNN.NN \le 999999999.99$

11. <u>Y</u>

Parameter that defines the position of the own ship or any contact in the Coordinate Grid System being used.

Its value is given in miles.

FORMAT: SNNNNNNNNNN.NN

where:

S--sign (+ or -)

N--N.NN--miles and hundredths of miles.

RANGE: $0.0 \leq$ NNNNNNNNNN.NN $\leq$ 9999999999.99

12. <u>Designation (Desig)</u>

Parameter that defines the name given to a particular contact. Consists of two alphabetic characters or one blank and one alphabetic character.

Needs to be unique for the sets of contacts maintained by the system at any given time. Upper or lower case characters can be used.

FORMAT: AA

where:

AA--any alphabetic character.

13. <u>Type</u>

Parameter that defines if the contact is of surface or sub-surface kind.

FORMAT: TT

where:

TT--can be SU (Surface), SS (Subsurface) or AA (Air)

14. <u>Class</u>

Parameter that defines the identity and purpose of any contact.

FORMAT: C

where

C-F if Friendly (FRI).

C-H if Hostile (HOS).

C-U if Unknown (UNK).

15. Scale

Parameter that defines the scale at which the picture presented at the plasma unit is displayed. Can be specified up to one hundredth of a mile/inch.

FORMAT: MM.MM

where:

MM.MM--mile and hundredths of miles per inch.

RANGE: $00.25 \leq MM.MM \leq 25.00$

16. Safe CPA Range

Parameter that defines the radius of a circle with center at the own ship; any contact that will pass through this security circle will be considered in collision.

FORMAT: YYYY

where:

YYYY--yards.

RANGE: $0050 \leq YYYY \leq 1000$

17. Wind Direction

This parameter indicates the true bearing from which the wind is blowing.

FORMAT: DDD.D

where:

DDD.D--degrees and tenths of degrees.

RANGE: $000.0 \leq DDD.D \leq 359.9$

18. Wind Speed:

This parameter indicates the speed at which the wind is blowing.

FORMAT: KK.K

where:

KK.K--knots and tenths of knots.

RANGE: $00.0 \leq KKK.K \leq 99.9$.

B. INPUT COMMAND

1. Origin Update

This command is used to modify the Coordinate Grid Origin parameters. It causes the system to change all the X/Y values that had been calculated, and also to redraw the picture represented at the plasma display with the last position of the own ship at the center.

It requires new latitude and longitude values.

2. Own Ship Update

This command is used to modify the parameters of the own ship: Latitude, Longitude, Course and Speed, in a selective way.

3. Create

This command is issued to record a new contact. Contact designation, type, class, bearing and range required.

4. Remove Contact

This command is used to remove a contact from the system.

5. Redesignate

This command is used to give a new Designation to any contact already in the system.

6. Contact Update

This command is used to update the information about any contact being maintained by the system: Type, Class, Bearing, Range, Course and Speed, may be changed selectively.

Designation of the contact and its parameters are desired to be updated.

7. Swap Contacts

This command is used to change the list of contacts that are being displayed on the status board.

Designations of contacts are those to be in and out of the display.

8. Time

This command is used to update/change all the parameters that the system has with respect to time: Time Zone Number, System Clock value and Time between updates.

9. CPA Safe Range Update

This command is used to update/change the value of the CPA safe Range parameter.

Remember that the initial default value of the CPA Safe Range is 0050 yards.

10. Wind Update

This command is used to introduce/update information about the wind. It requires new wind direction and speed values.

11. Scale Update

This command is used to modify the value of the scale parameter being used to define the window that limits the picture to be represented at the plasma display.

12. Plasma Reorient

This command is used to redefine the position of the window used to form the picture to be represented at the Plasma Display.

13. Reorient

This command is used to display the values of the Coordinate Grid Origin: Latitude and Longitude.

14. Scale

This command is used to display the value of the Scale parameter currently in use.

15. Own Ship

This command is used to display the parameters associated with last position of the own ship. First page is Positional data and second page is tactical data.

16. Contact Information

This command is used to display information about any contact being maintained by the system. It requires designation of contact whose information is desired.

17. Contacts in System

This command is used to obtain information about the designation of all the contacts in the system.

18. Request CPA Safe Range

This command is used to obtain information about the current value of the CPA Safe range.

19. Wind

This command is used to obtain information about the wind.

20. Time Between Updates

This command is used to obtain information about the current value of the Time between Updates parameter.

APPENDIX G

PROGRAM LISTINGS

2. CRT1

   CRT$MASTER$CLEAR
   SET$LOW$HOME
   CLEAR$LOW$SCREEN
   SET$HIGH$HOME
   PUT$SPACE
   PUT$TAB
   PUT$FS
   PUT$LF
   START$PROT$FIELD
   START$BLINK
   STOP$PROT$FIELD
   INTERP
   INIT$HIGH$SCREEN
   PRINT$TIME$ZONE
   PRINT$TIME
   PRINT$CAT$LONG
   PRINT$COURSE
   PRINT$SPEED
   PRINT$CONTACTS
   PRINT$MODE
   PRINT$CONTACT$INFO

3. FLT$AXCII

   ASCII$TO$FLOAT
   FRAC$TO$ASCII
   FLOAT$TO$ASCII

4. FLOATING$POINT

   INIT$FP
   ADJUST$OP
   ADJUSTI$OP
   VAL$RESULT
   VAL$RESULT$1
   VAL$RESULT$2
   COMPARE
   FLOAT$MSG$ERROR

CHECK
MUL
DIV
EDIV
FMUL
FDIV
FADD
FSUB
FSGR
FLTDS
FIXSD
FSGRT
FCMPR
FZTST
EXCH
COS$SIN
ARC$TAN

5. COMMANDS1
PRINT$ERROR$MSG
CHECK$YES$NO
CHECK$FP$VALUE
CHECK$INPUT
GET$DEGREES
GET$MINUTES
GET$SIGN
FP$FORMAT
RANGE$FORMAT
LAT$LONG$FORMAT
GRT$TIME$ZONE
GET$LAT
GET$LONG
GET$COURSE$BRG
GET$SPEED
GET$RANGE
GET$DESIG

GET$TYPE
GET$KIND
GET$SCALE

6. TIME
CLOCK
INITIATE$TIME
INITIATE$CLOCK
ACTUAL$TIME

7. DISPLAY$CMDS
CONV$LAT$LONG
DISPLAY$DESIG
DISPLAY$$TYPE
DISPLAY$CLASS
DISPLAY$LAT$LONG
DISPLAY$XY
DISPLAY$CRS$BRG
DISPLAY$SPD
DISPLAY$RANGE
DISPLAY$TIME
DISPLAY$ORIGIN
DISPLAY$SCALE
DISPLAY$OWN$SHIP
DISPLAY$CONTACT$INFO
DISPLAY$SYSTEM
DISPLAY$SAFE$RANGE
DISPLAY$WIND
DISPLAY$UPDATE$TIME

8. CPAS$MODULE
CONV$CONTACT$TIME
COPA$TIME$CONV
CONTACT$CRS$SPD
CPA$CALCULATION
GET$CPA

9. PLASMA$MODULE
   SET$WINDOW
   CLEAR$STRUCTURES
   DRAW$FRIENDLY$SYMBOL
   DRAW$UNK$HOS$SYMBOL
   DRAW$OWN$SHIP$SYMBOL
   CHECK$PLASMA
   NORMALIZE
   PUT$OS$CENTER
   PUT$CONTACT$CENTER
   FIXED$REORIENTATION
   PLASMA$REDESIG
   PLASMA$DELETE
   PLASMA$CONTACT
   PLASMA$OS
   DRAW$EVERYTHING
   DISPLAY$PLASMA$SCALE
   REORIENT$PS

10. PLASMA$PRIMITIVES
   SET$STATUS$PLASMA
   PLASMA$WRITE
   CLEAR$PLASMA
   PLASMA$WRITE$VECTOR
   PLASMA$PRINT$STRING
   INITIALIZE$PLASMA
   SET$VECTOR
   START$VECTOR$SOLID
   STOP$VECTOR$SOLID
   START$VECTOR$DASH
   STOP$VECTOR$DASH
   GRAPHIC$DESIG

11. EXECUTIVE$COMMANDS
   DE$HASH
   CHECK$GO$KEY
   DISPLAY$KIND

CHECK$DESIG
CONV$MIN$IRAD
CONV$RAD$MIN
CONV$XY
CONV$RCL$XY
INIT$STRUSTURES
GET$SYSTEM$PARAMETERS
DISPLAY$CONTACT
CREATE
REMOVE
REDESIGNATE
UPDATE
SWAP$CONTACTS
TRANSLATE
OWN$SHIP$UPDATE
ORIGIN
WIND
SCALE
GET$SAFE$RANGE
INPUT$TIME

12. ANAMODULE
NO$WIND
NO$CONTACT
NOT$ENOUGH$CONTACTS
TOO$MANY$CONTACTS
MOVE$OWN$SHIP
EXECUTIVE

13. AAW$GFCS
BINARY$TO$FLOAT
BINARY$TO$ASCII
FLOAT$TO$BINARY
SCALE
GEN$TRG$UALI
GEN$TRG$VALII
GEN$TRG$VALIII

GEN$TRG$VALIV
TO$GET$TF
PREDICTED$TRG$VAL
INV$SCALE
ITERATION

14. SW$GFCS

BINARY$TO$ASCII
FLOAT$TO$BINARY
SW$GEN$TRG$VALI
SW$GEN$TRG$VALII
SW$GEN$TRG$VALIII
SW$GEN$TRG$VALIV
SW$PRED$TRG$VAL

```
/****     SBC IN MDS SYSTEM     ****/

SBCMDS:
  DO;
    DECLARE LIT LITERALLY 'LITERALLY',
            DCL LIT 'DECLARE';

    DCL ATOD (18) BYTE AT(0F700H),
        BUFFER (18) BYTE,
        I BYTE,
        CP1 LIT '0E6H',
        CP2 LIT '0EAH',
        DATAPORTA1 LIT '0E4H',
        DATAPORTA2 LIT '0E8H',
        RDY LIT '0FH',
        NRDY LIT '0H',
        CONTRPRT1 LIT '0E7H',
        CONTRPRT2 LIT '0EBH';

    SIGNAL: PROCEDURE(CP, VAL);
        DCL (CP, VAL) BYTE;             /* CP-CONTROL PORT */
        OUTPUT (CP) = VAL;          /* VAL=0 NOT READY, 0FH READY*/
    END SIGNAL;

    WAIT: PROCEDURE(CP);
        DCL CP BYTE;                /* CP-CONTROL PORT */
        DO WHILE(INPUT(CP) = 0);    /* WAIT UNTIL READY */
        END;
    END WAIT;
```

```
DELAY: PROCEDURE;                        /* SHORT DELAY OF 28 STATES */

   END DELAY;

INITIALIZE: OUTPUT(CONTRPRT1) = 9BH; /* A1-C1 INPUT MODE Ø */
            OUTPUT(CONTRPRT2) = 80H; /* A2-C2 OUTPUT MODE Ø */

MAIN: DO WHILE 1;          /* DO FOREVER */

/*   READ D TO A VALUES INTO BUFFER */

DO I = 15 TO 17;
   BUFFER(I) = ATOD(I);
END;


/*****************************************************************
*
*  RECEIVE A TO D VALUES AND TRANSMIT D TO A VALUES TO
*  A TO D CONVERTER.
*
*****************************************************************/
   DO I = Ø TO 15;
      CALL WAIT(CP1);  /* WAIT FOR SIGNAL WHICH INDICATES THAT
                          DATAPORT CONTAINS DATA */
      BUFFER1(I) = INPUT(DATAPORTA1);     /* GET DATA VALUE */
      CALL SIGNAL(CP2, RDY); /* ACKNOWLEDGE RECEIPT OF DATA */
      CALL DELAY;            /* ALLOW SAFFICIENT TIME FOR
SIGNAL CP2 READY TO BE */
      CALL SIGNAL(CP2, NRDY); /* PICKED UP */
   END;
   DO I = 15 TO 17;
      OUTPUT(DATAPORTA2) = BUFFER(I); /* PLACE DATA IN PORT A2 */
      CALL SIGNAL(CP2, RDY);
      CALL WAIT (CP1);                /* WAIT FOR ACKNOWLEDGE */
      CALL SIGNAL(CP2, NRDY);
   END;

   DO I = 0 TO 15 ;       /* PLACE RECEIVED VALUES INTO MEMORY
      LOCATIONS ACCESSIBLE TO THE MDS SYSTEM

  */
      ATOD(I) = BUFFER(I);
   END;
  END;    /*  DO FOREVER   */

 END SBCMDS;    /*   END OF MODULE   */
```

```
/***   ADC-DAC CONTROLLER PROGRAM   ***/

ADC$DAC:
  DO;
     DECLARE LIT LITERALLY 'LITERALLY',
             DCL LIT 'DECLARE';

     DCL ATOD (18) BYTE AT (ØF7ØØH),
         BUFFER (18) BYTE, I BYTE,
         CP1 LIT 'ØE6H',
         CP2 LIT 'ØEAH',
         DATAPORTA1 LIT 'ØE4H',
         DATAPORTA2 LIT 'ØE8H',
         RDY LIT 'ØFH',
   NRDY LIT 'ØH',
         CONTRPRT1 LIT 'ØE7H',
         CONTRPRT2 LIT 'ØEBH';

  SIGNAL: PROCEDURE(CP, VAL);
    DCL (CP, VAL) BYTE;           /* CP - CONTROL PORT */
    OUTPUT(CP) = VAL;             /* VAL = Ø NOT READY, ØFH REALY */
  END SIGNAL;

  WAIT: PROCEDURE(CP, VAL);
    DCL (CP, VAL) BYTE;           /* CP - CONTROL PORT */
    DO WHILE(INPUT(CP) = Ø);      /* WAIT UNTIL READY */
    END;
  END WAIT;
```

```
 DELAY: PROCEDURE;
                                /* SHORT DELAY OF 28 STATES */
 END DELAY;

/***   PARALLEL PORTS ARE PROGRAMMED   ***/

INITIALIZE: OUTPUT(CONTPRT1) = 80H; /* A1-C1 OUTPUT MODE 0 */
            OUTPUT(CONTPRT2) = 9BH; /* A2-C2 INPUT MODE 0 */

MAIN: DO WHILE 1;               /* DO FOREVER */
      DO I = 0 TO 15;
        BUFFER(I) = ATOD(I);    /* GET ATOD VALUES */
      END;

/*****************************************************************
*
*  TRANSMIT A TO D VALUES TO MDS SYSTEM AND RECEIVE D TO A
*  VALUES FROM MDS.
*
*****************************************************************/

      DO I = Ø TO 15;
        OUTPUT(DATAPORTA1) = BUFFER(I);
        CALL SIGNAL (CP1, RDY);
        CALL WAIT (CP2);
        CALL SIGNAL (CP1, NRDY);
      END;
```

```
   DO I = 16 TO 17;
     CALL WAIT (CP2);          /* WAIT FOR A SIGNAL FROM MDS
 FOR DATA IN DATAPORTA2 */
     BUFFER(I) = INPUT(DATAPORTA2);
     CALL SIGNAL(CP1, RDY);    /* ACKNOWLEDGE DATA */
     CALL DELAY;
     CALL SIGNAL(CP1, NRDY);
   END;

   DO I = 16 TO 17;
     ATOD(I) = BUFFER(I);       /* PLACE RECEIVED VALUES INTO
  D TO A CONVERTER */
   END;

END ADC$DAC;
```

```
          /***    EXTER     ***/

DECLARE
   TIME$BUFFER(6) BYTE EXTERNAL,
   SEC$TABLE(8) BYTE EXTERNAL,
   (MILLI$SEC,DUMMY$SEC,SECONDS,MINUTES,HOURS,DAY,SEC$TIME)
   BYTE EXTERNAL,
   TIME$STEP ADDRESS EXTERNAL,
   MEAS ADDRESS EXTERNAL;

CRT$WRITE:
   PROCEDURE (CHAR) EXTERNAL;
   DECLARE CHAR BYTE; END;

CRT$PRINT$STRING:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

CRT$READ:
   PROCEDURE BYTE EXTERNAL;
   END;

CRT$TRY$READ:
   PROCEDURE BYTE EXTERNAL;
   END;

ECHO$CRT:
   PROCEDURE BYTE EXTERNAL;
   END;

SEND$SUB:
   PROCEDURE EXTERNAL;
   END;

SEND$CR:
   PROCEDURE EXTERNAL;
   END;

SEND$LF:
   PROCEDURE EXTERNAL;
   END;

SEND$CRLF:
   PROCEDURE EXTERNAL;
   END;

SEND$BEL:
   PROCEDURE EXTERNAL; END;

SEND$ES:
   PROCEDURE EXTERNAL; END;
```

```
SEND$SPACE:
   PROCEDURE (NUM) EXTERNAL;
   DECLARE NUM BYTE; END;

BYTE$CHAR:
   PROCEDURE (CHAR) EXTERNAL;
   DECLARE CHAR BYTE; END;

ADDRESS$CHAR:
   PROCEDURE (CHAR) EXTERNAL;
   DECLARE CHAR ADDRESS; END;

BYTES$TO$ASCII:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

GET$BYTE:
   PROCEDURE (A) BYTE EXTERNAL;
   DECLARE A BYTE; END;

GET$ADDRESS:
   PROCEDURE (A) ADDRESS EXTERNAL;
   DECLARE A BYTE; END;

GET$STRING:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE A ADDRESS, B BYTE; END;

PUT$NUMBER$BUFFER:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE A BYTE, B ADDRESS; END;

INIT$FP:
   PROCEDURE EXTERNAL;
   END;

MUL:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

DIV:
   PROCEDURE (A,B,C,D) EXTERNAL;
   DECLARE (A,B,C,D) ADDRESS; END;

EDIV:
   PROCEDURE (A,B,C,D) EXTERNAL;
   DECLARE (A,B,C,D) ADDRESS; END;

FMUL:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;
```

```
FDIV:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

FADD:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

FSUB:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

FSQR:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

FSQRT:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

FITDS:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

FIXSD:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

FCMPR:
   PROCEDURE (A,B,C) BYTE EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

FZTST:
   PROCEDURE (A,B) BYTE EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

FXCH:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

COS$SIN:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

ARC$TAN:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;
```

```
ASCII$TO$FLOAT:
   PROCEDURE (A,N,B) EXTERNAL;
   DECLARE (A,B) ADDRESS, N BYTE; END;

FLOAT$TO$ASCII:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

INITIATE$TIME:
   PROCEDURE EXTERNAL;
   END;

INITIATE$CLOCK:
   PROCEDURE EXTERNAL;
   END;

ACTUAL$TIME:
   PROCEDURE EXTERNAL;
   END;

SET$MASTER$CLEAR:
   PROCEDURE EXTERNAL;
   END;

SET$LOW$HOME:
   PROCEDURE EXTERNAL;
   END;

CLEAR$LOW$SCREEN:
   PROCEDURE EXTERNAL;
   END;

SET$HIGH$HOME:
   PROCEDURE EXTERNAL;
   END;

INIT$HIGH$SCREEN:
   PROCEDURE EXTERNAL;
   END;

START$BLINK:
   PROCEDURE EXTERNAL;
   END;

PRINT$TIME$ZONE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PRINT$TIME:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;
```

```
PRINT$LAT$LONG:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

PRINT$COURSE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PRINT$SPEED:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PRINT$CONTACTS:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PRINT$MODE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PRINT$CONTACT$INFO:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE A BYTE, B ADDRESS; END;

CHECK$YES$NO:
   PROCEDURE BYTE EXTERNAL;
   END;

CHECK$FP$VALUE:
   PROCEDURE (A,B) BYTE EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

CHECK$INPUT:
   PROCEDURE BYTE EXTERNAL;
   END;

GET$DEGREES:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE A BYTE, B ADDRESS; END;

GET$MINUTES:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$SIGN:
   PROCEDURE (A,B) BYTE EXTERNAL;
   DECLARE (A,B) BYTE; END;

FP$FORMAT:
   PROCEDURE (A,B,C,D) BYTE  EXTERNAL;
   DECLARE (A,B) ADDRESS, (C,D) BYTE; END;
```

```
RANGE$FORMAT:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

LAT$LONG$FORMAT:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B) ADDRESS, C BYTE; END;

GET$TIME$ZONE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$LAT:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$LONG:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$COURSE$BRG:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE A BYTE, B ADDRESS; END;

GET$SPEED:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$RANGE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$DESIG:
   PROCEDURE ADDRESS EXTERNAL;
   END;

GET$TYPE:
   PROCEDURE BYTE EXTERNAL;
   END;

GET$KIND:
   PROCEDURE BYTE EXTERNAL;
   END;

GET$SCALE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;
```

```
        /***    EXTER1    ***/

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';

DCL SYSTEM STRUCTURE
          (LAT (4) BYTE,
           LONG (4) BYTE,
           SCALE (4) BYTE,
           WIND$DIR (4) BYTE,
           WIND$SPD (4) BYTE,
           NUM$ZONE (5) BYTE,
           CONTACT$KIND (3) BYTE,
           NUMCTS BYTE ) EXTERNAL,

    OWN$SHIP$INFO STRUCTURE
                 (LAT (4) BYTE,
                  LONG (4) BYTE,
                  POINTER BYTE,
                  FLAG BYTE) EXTERNAL,

    OWN$SHIP (30) STRUCTURE
                 (X (4) BYTE,
                  Y (4) BYTE,
                  TIME (3) BYTE,
                  CRS (4) BYTE,
                  SPD (4) BYTE) EXTERNAL,

    CONTACT$INFO (15) STRUCTURE
                     (DESIG ADDRESS,
                      TYPE BYTE,
                      KIND BYTE,
                      CRS$FLAG BYTE,
                      SPD$FLAG BYTE,
                      OS$POINTER BYTE,
                      POINTER BYTE,
                      FLAG BYTE) EXTERNAL,

    CONTACT$POSI (225) STRUCTURE
                       (X (4) BYTE,
                        Y (4) BYTE,
                        TIME (3) BYTE,
                        CRS (4) BYTE,
                        SPD (4) BYTE,
                        BRG (4) BYTE,
                        RNG (4) BYTE) EXTERNAL;

DCL CONTACT$DISPLAY (6) BYTE EXTERNAL;
```

```
DCL
    LAT$STRING (9) BYTE EXTERNAL,
    LONG$STRING (9) BYTE EXTERNAL,
    CRS$STRING (6) BYTE EXTERNAL,
    SPD$STRING (5) BYTE EXTERNAL,
    CONTACTS$STRING (5) BYTE EXTERNAL,
    CONTACT$INFO$STRING (44) BYTE EXTERNAL;


DES$HASH:
    PROCEDURE (A,B) EXTERNAL;
    DCL (A,B) ADDRESS; END;

CHECK$GO$KEY:
    PROCEDURE EXTERNAL;
    END;

CONV$MIN$RAD:
    PROCEDURE (A,B) EXTERNAL;
    DCL (A,B) ADDRESS; END;

DISPLAY$KIND:
    PROCEDURE EXTERNAL;
    END;

CHECK$DESIG:
    PROCEDURE (A) BYTE EXTERNAL;
    DCL A ADDRESS; END;

CONV$RAD$MIN:
    PROCEDURE (A,B) EXTERNAL;
    DCL (A,B) ADDRESS; END;

CONV$XY:
    PROCEDURE (A,B,C,D) EXTERNAL;
    DCL (A,B,C,D) ADDRESS; END;

CONV$REL$XY:
    PROCEDURE (A,B,C,D) EXTERNAL;
    DCL (A,B,C,D) ADDRESS; END;

GET$SYSTEM$PARAMETERS:
    PROCEDURE EXTERNAL;
    END;

DISPLAY$CONTACT:
    PROCEDURE (A,B) EXTERNAL;
    DCL (A,B) BYTE; END;
```

```
CREATE:
   PROCEDURE EXTERNAL;
   END;

REMOVE:
   PROCEDURE EXTERNAL;
   END;

REDESIGNATE:
   PROCEDURE EXTERNAL;
   END;

UPDATE:
   PROCEDURE EXTERNAL;
   END;

SWAP$CONTACTS:
   PROCEDURE EXTERNAL;
   END;

OWN$SHIP$UPDATE:
   PROCEDURE EXTERNAL;
   END;

ORIGIN:
   PROCEDURE EXTERNAL;
   END;

WIND:
   PROCEDURE EXTERNAL;
   END;

SCALE:
   PROCEDURE EXTERNAL;
   END;

INPUT$TIME:
   PROCEDURE BYTE EXTERNAL;
   END;
```

```
        /***    EXT[illegible]    ***/

GET$STATUS$PLASMA:
   PROCEDURE (A) EXTERNAL;
   DECLARE A BYTE; END;

PLASMA$WRITE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A BYTE; END;

CLEAR$PLASMA:
   PROCEDURE EXTERNAL;
   END;

PLASMA$WRITE$VECTOR:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PLASMA$PRINT$STRING:
   PROCEDURE (A, B, C) EXTERNAL;
   DECLARE (A, B) BYTE, C ADDRESS; END;

INITIALIZE$PLASMA:
   PROCEDURE EXTERNAL;
   END;

SET$VECTOR:
   PROCEDURE (A, B, C) EXTERNAL;
   DECLARE (A, B, C) ADDRESS; END;

START$VECTOR$SOLID:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS; END;

STOP$VECTOR$SOLID:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS; END;

START$VECTOR$DASH:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS; END;

STOP$VECTOR$DASH:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS; END;

GRAPHIC$LESIG:
   PROCEDURE (A, B, C) EXTERNAL;
   DECLARE (A, B, C) ADDRESS; END;
```

```
PL/M-80 COMPILER                                          20 MAR 81  PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE BASICS
OBJECT MODULE PLACED IN BASICS.OBJ
COMPILER INVOKED BY:  :F1:PLM80 BASICS.SRC PAGELENGTH(33) PAGEWIDTH(75) DAT
                     -E(20 MAR 81)



   1          BASICS: DO;

   2   1      DECLARE LIT LITERALLY 'LITERALLY',
                    DCL LIT 'DECLARE';
   3   1      DCL CR LIT '0DH',                  /* CARRIAGE RETURN.*/
                  LF LIT '0AH',                  /* LINE FEED.*/
                  EOL LIT '24H',                 /* $$ INDICATES END OF LINE.
         -  */
                  BS LIT '08H',                  /* BACKSPACE. */
                  SPACE LIT '20H',               /* SPACE */
                  RUB LIT '7FH',                 /* RUB OUT. */
                  SUB LIT '1AH',                 /* UP ROW CURSOR. */
                  BEL LIT '07H';                 /* RING THE BELL. */
   4   1      DCL TRUE LIT '0FFH',
                  FALSE LIT '00H',
                  FOREVER LIT 'WHILE TRUE';
   5   1      DCL CRTDATA LIT '0F6H',            /* CRT DATA ON PORT 246.*/
                  CRTSTATUS LIT '0F7H',          /* CRT STATUS ON PORT 247.*/
                  CRT$REC$MASK LIT '06H',        /* MASK FOR CRT: RECEIVE.*/
                  CRT$TRT$MASK LIT '05H';        /* MASK FOR CRT: TRANSMIT. *
         -  /
```

```
                 /*****************************************************************
               -  *********
                 *
                 * CRT$WRITE:
                 *   PROCEDURE USED TO SEND A CHARACTER TO THE CRT.
                 *
                 * PARAMETERS:
                 *    - CHAR.-CHARACTER BYTE TO BE SENT.
                 *
                 ******************************************************************
               -  ********/
   6    1        CRT$WRITE: PROCEDURE (CHAR) PUBLIC;
   7    2           DCL CHAR BYTE;
   8    2           DO WHILE (INPUT(CRTSTATUS) AND CRT$TRT$MASK) <> CRT$TRT$MA
               -  SK;  /*WAIT*/
   9    3              END;
  1Ø    2           OUTPUT(CRTDATA) = CHAR;
  11    2           END CRT$WRITE;


                 /*****************************************************************
               -  ********
                 *
                 * CRT$PRINT$STRING:
                 *   PROCEDURE USED TO SEND A STRING OF CHARACTERS TO THE CRT.
                 *
                 * PARAMETERS:
                 *    - A.-POINTER TO STRING. $$ WILL INDICATE END OF STRING. M
               -  AXIMUM
```

```
                  *       LENGTH: 80 CHARACTERS.
                  *
                  ****************************************************************
              -    ********/
 12   1           CRT$PRINT$STRING: PROCEDURE (A) PUBLIC;
 13   2              DCL (POINTER,A) ADDRESS,
                         (BUFFER BASED A) (80) BYTE;
 14   2              POINTER = 0;
 15   2              DO WHILE (BUFFER(POINTER) <> EOL) OR (BUFFER(POINTER + 1)
              -    <> EOL);
 16   3                 CALL CRT$WRITE(BUFFER(POINTER));
 17   3                 POINTER = POINTER + 1;
 18   3                 END;
 19   2              END CRT$PRINT$STRING;

                  /***************************************************************
              -    *******
                  *
                  * CRT$READ:
                  *    PROCEDURE USED TO RECEIVE A CHARACTER FROM THE CRT, AND R
              -    ETURN ITS
                  *    VALUE TO THE CALLING MODULE.
                  *
                  * USAGE:
                  *    UNTYPED PROCEDURE. RETURNS A BYTE VALUE. (ASCII CODE)
                  *
                  ****************************************************************
              -    *******/
 20   1           CRT$READ: PROCEDURE BYTE PUBLIC;
 21   2              DCL CHAR BYTE;
 22   2          DO WHILE (INPUT(CRTSTATUS) AND CRT$REC$MASK) <> CRT$REC$MA
            -   SK; /*WAIT*/
 23   3            END;
 24   2          CHAR = INPUT(CRTDATA);
 25   2          IF CHAR >= 80H THEN CHAR = CHAR XOR 80H;
 27   2          RETURN CHAR;
 28   2          END CRT$READ;


               /*************************************************************
            -   *******
               *
               * CRT$TRY$READ:
               *   PROCEDURE USED TO TEST IF CHARACTER HAS BEEN SENT FROM CR
            -   T.
               *
               * USAGE:
               *   UNTYPED PROCEDURE. IF READING WAS SUCCESSFUL, THE ASCII C
            -   HARACTER
               *   IS DISPLAYED AND RETURNS ITS BYTE VALUE TO THE CALLING MO
            -   DULE.
               *   IF READING WAS NOT SUCCESSFUL, THEN A NUL ASCII CHARACTER
            -    (00H) IS
               *   RETURNED, AND NO CHARACTER IS DISPLAYED.
               *
               **************************************************************
            -   ******/
 29   1        CRT$TRY$READ: PROCEDURE BYTE PUBLIC;
 30   2          DCL CHAR BYTE;
 31   2          CHAR = 00H;
 32   2          IF (INPUT(CRTSTATUS) AND CRT$REC$MASK) = CRT$REC$MASK
                    THEN CHAR = INPUT(CRTDATA);
 34   2          IF CHAR >= 80H THEN CHAR = CHAR XOR 80H;
 36   2          RETURN CHAR;
 37   2          END CRT$TRY$READ;


               /************************************************************
             -  *****
               *
               * ECHO$CRT:
               *   PROCEDURE USED TO RECEIVE AN ASCII CHARACTER FROM THE CRT
             -  , DISPLAY
               *   THE SAME, AND RETURN ITS BYTE VALUE TO THE CALLING MODULE
             -  .
               *
               * USAGE:
               *   UNTYPED PROCEDURE. RETURNS A BYTE VALUE. (ASCII CODE.)
               *
               *************************************************************
             -  ****/
 38   1        ECHO$CRT: PROCEDURE BYTE PUBLIC;
 39   2          DCL CHAR BYTE;
 40   2          CHAR = CRT$READ;
 41   2          CALL CRT$WRITE(CHAR);
 42   2          RETURN CHAR;
 43   2          END ECHO$CRT;


               /************************************************************
```

```
              -   *****
                  *
                  * SEND$SUB:
                  *   PROCEDURE USED TO SEND AN UP ROW CURSOR CHARACTER TO THE
              -   CRT.
                  *
                  ************************************************************
              -   ******/
  44   1          SEND$SUB: PROCEDURE PUBLIC;
  45   2             CALL CRT$WRITE(SUB);
  46   2             END SEND$SUB;


                  /***********************************************************
              -   ****
                  *
                  * SEND$CR:
                  *   PROCEDURE USED TO SEND A CR ASCII CHARACTER TO THE CRT.
                  *
                  ************************************************************
              -   **/
  47   1          SEND$CR: PROCEDURE PUBLIC;
  48   2             CALL CRT$WRITE(CR);
  49   2             END SEND$CR;


                  /***********************************************************
              -   **
                  *
```

```
                * SEND$LF:
                *   PROCEDURE USED TO SEND A LF ASCII CHARACTER TO THE CRT.
                *
                ***************************************************************
              -  */
 50   1         SEND$LF: PROCEDURE PUBLIC;
 51   2            CALL CRT$WRITE(LF);
 52   2            END SEND$LF;

                /**************************************************************
              -  ***
                *
                * SEND$CRLF:
                *   PROCEDURE USED TO SEND BOTH CR AND LF ASCII CHARACTERS TO
              -  CRT.
                *
                ***************************************************************
              -  **/
 53   1         SEND$CRLF: PROCEDURE PUBLIC;
 54   2            CALL CRT$WRITE(CR);
 55   2            CALL CRT$WRITE(LF);
 56   2            END SEND$CRLF;

                /**************************************************************
              -  **
                *
                * SEND$BEL:
                *   PROCEDURE USED TO SEND A 'BELL' ASCII CHARACTER TO THE CR
```

```
                  -  T.
                     *
                     ***************************************************************
                  -  /
  57   1             SEND$BEL: PROCEDURE PUBLIC;
  58   2                CALL CRT$WRITE(BEL);
  59   2                END SEND$BEL;



                     /**************************************************************
                  -   ***
                     *
                     * SEND$BS:
                     *   PROCEDURE USED TO SEND A NON-DESTRUCTIVE BACKSPACE CHARAC
                  -   TER TO
                     *   THE CRT.
                     *
                     ***************************************************************
                  -   *****/
  60   1             SEND$BS: PROCEDURE PUBLIC;
  61   2                CALL CRT$WRITE(BS);
  62   2                END SEND$BS;

                     /**************************************************************
                  -   ********
                     *
                     * SEND$SPACE:
```

```
                    *   THIS PROCEDURE IS USED TO SEND SPACES TO THE CRT.
                    *
                    * PARAMETERS:
                    *   - NUM.- NUMBER OF SPACES DESIRED.
                    *
                    **************************************************************
                  -  *********/
 63   1             SEND$SPACE: PROCEDURE (NUM) PUBLIC;
 64   2                DCL NUM BYTE;
 65   2                DO WHILE NUM > 0;
 66   3                   CALL CRT$WRITE(SPACE);
 67   3                   NUM = NUM - 1;
 68   3                   END;
 69   2                END SEND$SPACE;


                    /*************************************************************
                    *
                    * BYTE$CHAR:
                    *   PROCEDURE USED TO DISPLAY THE DECIMAL VALUE OF A BYTE VAR
                  -  IA-
                    *   BLE.
                    *
                    * PARAMETERS:
                    *   - CHAR.- BYTE VALUE DESIRED TO BE DISPLAYED.
                    *
                    *************************************************************/
 70   1             BYTE$CHAR: PROCEDURE (CHAR) PUBLIC;
 71   2                DCL VALUE(3) BYTE DATA (100,10,1);
 72   2         DCL (I,CHAR,COUNT,TEMP) BYTE;
 73   2         DO I = 0 TO LAST(VALUE);
 74   3            COUNT = 0;
 75   3            TEMP = VALUE(I);
 76   3            DO WHILE CHAR >= TEMP;
 77   4               CHAR = CHAR - TEMP;
 78   4               COUNT = COUNT + 1;
 79   4               END;
 80   3            CALL CRT$WRITE(COUNT + '0');
 81   3            END;
 82   2         END BYTE$CHAR;


             /*************************************************************
             *
             * ADDRESS$CHAR:
             *   PROCEDURE USED TO DISPLAY THE DECIMAL VALUE OF AN ADDRESS
             *   VARIABLE.
             *
             * PARAMETERS:
             *   - CHAR.- ADDRESS VALUE DESIRED TO BE DISPLAYED.
             *
             *************************************************************
              /
 83   1      ADDRESS$CHAR: PROCEDURE (CHAR) PUBLIC;
 84   2         DCL VALUE(5) ADDRESS DATA (10000,1000,100,10,1);
 85   2         DCL (I,COUNT) BYTE,
                    (CHAR,TEMP) ADDRESS;
 86   2         DO I = 0 TO LAST(VALUE);
 87   3             COUNT = Ø;
 88   3             TEMP = VALUE(I);
 89   3             DO WHILE CHAR >= TEMP;
 90   4                CHAR = CHAR - TEMP;
 91   4                COUNT = COUNT + 1;
 92   4                END;
 93   3             CALL CRT$WRITE(COUNT + 'Ø');
 94   3             END;
 95   2          END ADDRESS$CHAR;


             /****************************************************************
          -   **********
              *
              * BYTE$TO$ASCII:
              *   THIS PROCEDURE IS USED TO CONVERT A BYTE QUANTITY INTO TW
          -   O ASCII
              *   CHARACTERS REPRESENTING ITS HEXADECIMAL VALUE.
              *
              * PARAMETERS:
              *   - A.- POINTER TO THE BYTE QUANTITY DESIRED TO BE CONVERTE
          -   D.
              *   - B,C.- POINTERS TO TWO BYTE QUANTITIES IN WHICH THE TWO
          -   ASCII
              *            CHARACTERS WILL BE PLACED. B POINTS TO THE MOST S
          -   IGNIFICANT
              *            PORTION OF THE ANSWER.
              *
              ****************************************************************
              -   *********/
 96    1          BYTE$TO$ASCII: PROCEDURE (A,B,C) PUBLIC;
 97    2             DCL (A,B,C) ADDRESS,
                         ENTRY BASED A BYTE,
                         OUT1 BASED B BYTE,
                         OUT2 BASED C BYTE,
                         TEMP BYTE;
 98    2             TEMP = SHR(ENTRY,4);
 99    2             IF TEMP <= 9 THEN OUT1 = TEMP + 30H;
101    2                          ELSE OUT1 = TEMP + 37H;
102    2             TEMP = ENTRY AND 00FH;
103    2             IF TEMP <= 9 THEN OUT2 = TEMP + 30H;
105    2                          ELSE OUT2 = TEMP + 37H;
106    2             END BYTE$TO$ASCII;


                  /*************************************************************
              -   ******
                  *
                  * GET$BYTE:
                  *   PROCEDURE USED TO OBTAIN A DECIMAL NUMBER BETWEEN 0 AND 2
              -   55 FROM
                  *   THE CRT.
                  *
                  * PARAMETERS:
                  *   - DIGITS.- INDICATES THE NUMBER OF DIGITS DESIRED.MUST BE
              -    LESS THAN
                  *     OR EQUAL TO 3, ALTHOUGH NO CHECK OF THIS IS MADE.
                  *
                  * USAGE:
                   *   TYPED PROCEDURE THAT WILL RETURN A DECIMAL DIGIT OBTAINED
                 -   FROM
                   *   THE CRT AND REPRESENTABLE IN 3 OR LESS DIGITS.
                   *   THE VALUE RETURNED WILL ALWAYS BE LESS THAN 255.
                   *
                   ***************************************************************
                 -  *******/
 107   1           GET$BYTE: PROCEDURE(DIGITS) BYTE PUBLIC;
 108   2              DCL (NUMBER,DIGITS,CHAR,COUNT) BYTE;
 109   2              NUMBER, COUNT = 0;
 110   2              DO WHILE DIGITS > 0;
 111   3                 CHAR = CRT$READ;
 112   3                 DO WHILE (((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> R
                 -  UB)) OR
                         ((CHAR = RUB) AND (COUNT = 0));
 113   4                    CALL SEND$BEL;
 114   4                    CHAR = CRT$READ;
 115   4                    END;
 116   3                 IF CHAR <> RUB
                         THEN DO;
 118   4                    CALL CRT$WRITE (CHAR);
 119   4                    NUMBER = NUMBER*10 + (CHAR - 30H);
 120   4                    COUNT = COUNT + 1;
 121   4                    DIGITS = DIGITS - 1;
 122   4                    END;
 123   3                 ELSE DO;
 124   4                    NUMBER = NUMBER/10;
 125   4                    CALL SEND$BS;
 126   4                    COUNT = COUNT - 1;
 127   4                    DIGITS = DIGITS + 1;
 128   4              END;
 129   3           END;
 130   2        RETURN NUMBER;
 131   2        END GET$BYTE;


                /****************************************************************
           -     ***************
                *
                * GET$ADDRESS:
                *   PROCEDURE USED TO OBTAIN A DECIMAL NUMBER BETWEEN 0 AND 6
           -     5535 FROM
                *   THE CRT.
                *
                * PARAMETERS:
                *   - DIGITS.- INDICATES THE NUMBER OF DIGITS DESIRED. MUST B
           -     E LESS THAN
                *     OR EQUAL TO 5, ALTHOUGH NO CHECK OF THIS IS MADE.
                *
                * USAGE:
                *   TYPED PROCEDURE THAT WILL RETURN A DECIMAL VALUE OBTAINED
           -      FROM THE
                *   CRT AND REPRESENTABLE IN 5 OR LESS DIGITS. THE VALUE RETU
           -     RNED WILL
                *   ALWAYS BE LESS THAN 65536.
                *
                *****************************************************************
           -     ****************/
 132   1        GET$ADDRESS: PROCEDURE(DIGITS) ADDRESS PUBLIC;
```

```
PL/M-80 COMPILER                                          20 MAR 81  PAGE  15

 133   2        DCL (CHAR,DIGITS,COUNT) BYTE,
                    NUMBER ADDRESS;
 134   2        NUMBER, COUNT = 0;
 135   2        DO WHILE DIGITS > 0;
 136   3           CHAR = CRT$READ;
 137   3           DO WHILE (((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> R
         -    UB)) OR
               ((CHAR = RUB) AND (COUNT = 0));
 138   4              CALL SEND$BEL;
 139   4              CHAR = CRT$READ;
 140   4              END;
 141   3           IF CHAR <> RUB
               THEN DO;
 143   4              CALL CRT$WRITE(CHAR);
 144   4              NUMBER = NUMBER*10 + (CHAR - 30H);
 145   4              COUNT = COUNT + 1;
 146   4              DIGITS = DIGITS - 1;
 147   4              END;
 148   3           ELSE DO;
 149   4              CALL SEND$BS;
 150   4              NUMBER = NUMBER/10;
 151   4              COUNT = COUNT - 1;
 152   4              DIGITS = DIGITS + 1;
 153   4              END;
 154   3           END;
 155   2        RETURN NUMBER;
 156   2        END GET$ADDRESS;
```

```
                  /**************************************************************
               -   ****************
                  *
                  * GET$STRING:
                  *   PROCEDURE USED TO OBTAIN A STRING OF 'NUMBER' CHARACTERS
               -   FROM THE
                  *   CRT.
                  *
                  * PARAMETERS:
                  *    - NUMBER.- NUMBER OF CHARACTERS DESIRED.
                  *    - A.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS
               -    DESIRED
                  *      TO BE PLACED.
                  *
                  * USAGE:
                  *   UNTYPED PROCEDURE. THE ONLY CHARACTERS THAT CAN BE ACCEPT
               -   ED FROM
                  *   THE CRT ARE: NUMBERS, UPPER AND LOWER CASE ALPHABETICS, A
               -   LTHOUGH
                  *   ALL LOWER CASE ALPHABETICS WILL BE CONVERTED INTO UPPER C
               -   ASE.
                  *
                  ***************************************************************
               -   ****************/
 157   1          GET$STRING: PROCEDURE(A,NUMBER) PUBLIC;
 158   2             DCL A ADDRESS,
                         (CHAR BASED A, TEMP, NUMBER,COUNT) BYTE;
 159   2             COUNT = 0;
 160   2             DO WHILE NUMBER > 0;
 161   3                CHAR = CRT$READ;
 162   3            DO WHILE (((CHAR < 30H) OR (CHAR > 7AH) OR ((CHAR > 39H
              -   ) AND (CHAR < 41H))
                    OR ((CHAR > 5AH) AND (CHAR < 61H))) AND (CHAR <> RUB))
              -   OR
                    ((CHAR = RUB) AND (COUNT = 0));
 163   4              CALL SEND$BEL;
 164   4              CHAR = CRT$READ;
 165   4              END;
 166   3            IF (CHAR >= 61H) AND (CHAR <= 7AH)
                      THEN CHAR = CHAR - 20H;
 168   3            IF CHAR <> RUB
                    THEN DO;
 170   4              CALL CRT$WRITE (CHAR);
 171   4              A = A + 1;
 172   4              NUMBER = NUMBER - 1;
 173   4              COUNT = COUNT + 1;
 174   4              END;
 175   3            ELSE DO;
 176   4              CALL SEND$BS;
 177   4              A = A - 1;
 178   4              NUMBER = NUMBER + 1;
 179   4              COUNT = COUNT - 1;
 180   4              END;
 181   3          END;
 182   2          END GET$STRING;


                  /*****************************************************************
```

```
             -   **************
                 *
                 * PUT$NUMBER$BUFFER:
                 *   THIS PROCEDURE IS USED TO OBTAIN AN SPECIFIED NUMBER OF N
             -   UMERIC CHARACTERS
                 *   FROM THE CRT, AND TO PUT THEM IN A GIVEN MEMORY LOCATION.
                 *
                 * PARAMETERS:
                 *   - NUM.- NUMBER OF NUMERIC CHARACTERS DESIRED.
                 *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS
             -    DESIRED TO
                 *          BE PLACED.
                 *
                 ****************************************************************
             -   ****************/
 183   1         PUT$NUMBER$BUFFER: PROCEDURE(NUM,A) PUBLIC;
 184   2            DCL A ADDRESS,
                          (TEMP BASED A, NUM, CHAR, COUNT) BYTE;
 185   2            COUNT = 0;
 186   2            DO WHILE NUM > 0;
 187   3               CHAR = CRT$READ;
 188   3               DO WHILE (((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> R
             -   UB)) OR
                       ((CHAR = RUB) AND (COUNT = 0));
 189   4                  CALL SEND$BEL;
 190   4                  CHAR = CRT$READ;
 191   4                  END;
 192   3               IF CHAR <> RUB
                       THEN DO;
 194   4                  CALL CRT$WRITE(CHAR);
```

```
PL/M-80 COMPILER                                          20 MAR 81  PAGE  19

 195   4              TEMP = CHAR;
 196   4              A = A + 1;
 197   4              NUM = NUM - 1;
 198   4              COUNT = COUNT + 1;
 199   4              END;
 200   3            ELSE DO;
 201   4              CALL SEND$BS;
 202   4              A = A - 1;
 203   4              NUM = NUM + 1;
 204   4              COUNT = COUNT - 1;
 205   4              END;
 206   3            END;
 207   2          END PUT$NUMBER$BUFFER;


 208   1        END BASICS;


MODULE INFORMATION:

     CODE AREA SIZE     = 04D6H   1238D
     VARIABLE AREA SIZE = 002DH     45D
     MAXIMUM STACK SIZE = 0004H      4D
     474 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER                                           20 MAR 81  PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE CRT
OBJECT MODULE PLACED IN CRT1.OBJ
COMPILER INVOKED BY:  :F1:PLM80 CRT1.SRC PAGELENGTH(33) PAGEWIDTH(75) DATE(
                   -20 MAR 81)


  1          CRT: DO;

  2   1      CRT$WRITE:
               PROCEDURE (A) EXTERNAL;
  3   2        DECLARE A BYTE; END;

  5   1      CRT$PRINT$STRING:
               PROCEDURE (A) EXTERNAL;
  6   2        DECLARE A ADDRESS; END;

  8   1      DECLARE
               LIT LITERALLY 'LITERALLY',
               DCL LIT 'DECLARE';

  9   1      DCL LF LIT 'ØAH',                   /* LINE FEED. */
                 CR LIT 'ØDH',                   /* CARRIAGE RETURN. */
                 MASTER$CLEAR LIT '1EH',         /* MASTER CLEAR. */
                 BLINK LIT 'ØEH',                /* BLINK ON. */
                 ETEOL LIT '17H',                /* ERASE TO END OF LINE
         -     . */
                 EOL LIT '24H',                  /* END OF LINE. */
```

```
            HOME LIT '02H',                          /* HOME. */
            TAB LIT '09H',                           /* TAB. */
            FS LIT '1CH',                            /* FORWARD CURSOR. NON-
 -        DESTRUCTIVE. */
            PROT$FIELD LIT '0FH',                    /* START PROTECTED FIEL
 -        D. */
            END$PF LIT '18H',                        /* STOP PROTECTED FIELD
 -        . */
            SPACE LIT '20H';                         /* SPACE. */

   12   1       DCL LIN1A (*) BYTE DATA
            ('  TIME: $2:$2:$2  LAT: $2:$2. $1 $1  LONG: $3:$2. $1 $1
 -          TIME ZONE: $3%9%3#'),
            LIN2A (*) BYTE DATA
            ('  OWNSHIP COURSE: $3. $1  SPEED: $2. $1%7$2F $2H $2J
 -         MODE: $7%6%1#'),
            LIN3A (*) BYTE DATA
            ('%9%9%9%3LAST  MARK%9%9%7C.P.A%9%1#'),
            LIN4A (*) BYTE DATA
            ('|CONTACT|QUAD|TYPE|CLASS| TIME | BRG | RANGE |COURSE|SP
 -        EED| TIME | BRG | RANGE |#'),
            LIN5A (*) BYTE DATA
            ('|-------|----|----|-----|------|-----|-------|------|--
 -        ---|------|-----|-------|#'),
            LIN6A (*) BYTE DATA
            ('|  $2   | $2 | $2 | $3 | $2:$2|$3.$1| $6| $3.$1| $2.$1|
 -         $2:$2|$3.$1| $6|#');
```

```
                /**************************************************************
           -     ****************
                *
                * CRT$MASTER$CLEAR:
                *    THIS PROCEDURE WILL CLEAR THE ENTIRE SCREEN, AND PUT THE
           -     CURSOR AT HOME.
                *
                ***************************************************************
           -     ****************/
 11    1        CRT$MASTER$CLEAR: PROCEDURE PUBLIC;
 12    2           CALL CRT$WRITE(MASTER$CLEAR);
 13    2           END CRT$MASTER$CLEAR;


                /**************************************************************
           -     ****************
                *
                * SET$LOW$HOME:
                *    THIS PROCEDURE WILL LOCATE THE CURSOR AT THE FIRST COLUMN
           -      IN ROW 17.
                *
                ***************************************************************
           -     ****************/
 14    1        SET$LOW$HOME: PROCEDURE PUBLIC;
 15    2           DCL I BYTE;
 16    2           CALL CRT$WRITE(HOME);
 17    2           DO I = 1 TO 16;
 18    3              CALL CRT$WRITE(CR);
 19    3              END;
```

```
  20    2        END SETSLOWSHOME;


                /***********************************************************
              -  *************
                *
                * CLEAR$LOW$SCREEN:
                *   THIS PROCEDURE WILL CLEAR ROWS 17 THRU 24.
                *   AFTER THIS OPERATION, THE CURSOR WILL BE PLACED AT COLUMN
              -   1 IN
                *   ROW 17.
                *
                ************************************************************
              -  ***********/
  21    1       CLEAR$LOW$SCREEN: PROCEDURE PUBLIC;
  22    2          DCL I BYTE;
  23    2          CALL SET$LOW$HOME;
  24    2          CALL CRT$WRITE(ETEOL);
  25    2          DO I = 1 TO 7;
  26    3             CALL CRT$WRITE(LF);
  27    3             CALL CRT$WRITE(ETEOL);
  28    3             END;
  29    2          CALL SET$LOW$HOME;
  30    2          END CLEAR$LOW$SCREEN;


                /***********************************************************
              -  *************
```

```
                 *
                 * SET$HIGH$HOME:
                 *   THIS PROCEDURE WILL LOCATE THE CURSOR AT THE COLUMN 1 AT
               -  ROW 1.
                 *
                 *************************************************************
               -  ************/
 31   1          SET$HIGH$HOME: PROCEDURE PUBLIC;
 32   2            CALL CRT$WRITE(HOME);
 33   2            END SET$HIGH$HOME;


                 /*************************************************************
               -  ************
                 *
                 * PUT$SPACE:
                 *   THIS PROCEDURE WILL WRITE A CERTAIN NUMBER OF SPACES AT T
               -  HE CRT.
                 *
                 * PARAMETERS:
                 *   - NUM.- NUMBER OF SPACES TO BE WRITTEN.
                 *
                 *************************************************************
               -  ************/
 34   1          PUT$SPACE: PROCEDURE(NUM);
 35   2            DCL NUM BYTE;
 36   2            DO WHILE NUM > 0;
 37   3              CALL CRT$WRITE(SPACE);
 38   3              NUM = NUM - 1;
 39   3            END;
 40   2          END PUT$SPACE;


                /************************************************************
            -    ***************
                *
                * PUT$TAB:
                *    THIS PROCEDURE WILL SEND A GIVEN NUMBER OF 'TABS' TO THE
            -    CRT.
                *
                * PARAMETERS:
                *    - NUM.- NUMBER OF TABS DESIRED.
                *
                *************************************************************
            -    ***************/
 41   1         PUT$TAB: PROCEDURE(NUM);
 42   2            DCL NUM BYTE;
 43   2            DO WHILE NUM > 0;
 44   3               CALL CRT$WRITE(TAB);
 45   3               NUM = NUM - 1;
 46   3               END;
 47   2            END PUT$TAB;


                /************************************************************
            -    ***************
                *
```

```
                  * PUTSFS:
                  *   THIS PROCEDURE WILL SEND TO THE CRT A GIVEN NUMBER OF NON
               -   -DESTRUCTIVE
                  *   FORWARD CURSOR CHARACTERS.
                  *
                  * PARAMETERS:
                  *   - NUM.- NUMBER OF NON-DESTRUCTIVE FORWARD CURSOR CHARACTE
               -   RS DESIRED.
                  *
                  ***************************************************************
               -   ***************/
  48   1          PUTSFS: PROCEDURE (NUM);
  49   2             DCL NUM BYTE;
  50   2             DO WHILE NUM > 0;
  51   3                CALL CRT$WRITE(FS);
  52   3                NUM = NUM - 1;
  53   3                END;
  54   2             END PUT$FS;


                  /**************************************************************
               -   ***************
                  *
                  * PUT$LF:
                  *   THIS PROCEDURE WILL SEND AN SPECIFIED NUMBER OF LINE FEED
               -    CHARACTERS
                  *   TO THE CRT.
                  *
                  * PARAMETERS:
```

```
                  *   - NUM.- NUMBER OF LINE FEED CHARACTERS DESIRED.
                  *
                  *****************************************************************
                -  ***************/
 55   1           PUT$LF: PROCEDURE(NUM);
 56   2              DCL NUM BYTE;
 57   2              DO WHILE NUM > 0;
 58   3                 CALL CRT$WRITE(LF);
 59   3                 NUM = NUM - 1;
 60   3                 END;
 61   2              END PUT$LF;


                  /*****************************************************************
                -  ******
                  *
                  * PUT$CR:
                  *   THIS PROCEDURE WILL SEND AN SPECIFIED NUMBER OF CARRIAGE
                -  RETURN
                  *   CHARECTERS TO THE CRT.
                  *
                  * PARAMETERS:
                  *   - NUM.- NUMBER OF CARRIAGE RETURN CHARECTERS DESIRED.
                  *
                  *****************************************************************
                -  *****/
 62   1           PUT$CR: PROCEDURE(NUM);
 63   2              DCL NUM BYTE;
 64   2              DO WHILE NUM > 0;
 55   3            CALL CRT$WRITE(CR);
 66   3            NUM = NUM - 1;
 67   3            END;
 68   2         END PUT$CR;


                /************************************************************
            -    ***************
                *
                * START$PROT$FIELD:
                *   THIS PROCEDURE WILL CAUSE ALL CHARACTERS SENT AFTER IT FI
            -    NISHES, TO
                *   BE IN A PROTECTED FIELD.
                *
                *************************************************************
            -    ***********/
 69   1         START$PROT$FIELD: PROCEDURE;
 70   2            CALL CRT$WRITE(PROT$FIELD);
 71   2            END START$PROT$FIELD;


                /************************************************************
            -    ***************
                *
                * START$BLINK:
                *   THIS PROCEDURE WILI CAUSE ALL CHARACTERS SENT AFTER IT TO
            -     BLINK.
                *
```

```
                 ************************************************************
               -  ****************/
 72   1          START$BLINK: PROCEDURE PUBLIC;
 73   2            CALL CRT$WRITE(BLINK);
 74   2            END START$BLINK;


                 /***********************************************************
               -  *************
                 *
                 * STOP$PROT$FIELD:
                 *   THIS PROCEDURE WILL CAUSE THE END TO 'INSERT PROTECTED FI
               -  ELD',
                 *   'ROLL MODE' AND 'BLINK ON'.
                 *
                 ************************************************************
               -  *************/
 75   1          STOP$PROT$FIELD: PROCEDURE;
 76   2            CALL CRT$WRITE(END$PF);
 77   2            END STOP$PROT$FIELD;


                 /***********************************************************
               -  *********
                 *
                 * INTERP:
                 *   THIS PROCEDURE IS USED BY 'INIT$HIGH$SCREEN' TO INTERPRET
               -   THE
```

```
PL/M-8Ø COMPILER                                              2Ø MAR 81   PAGE  11

                  *   STREAM OF CHARACTERS USED TO INITIALIZE THE HIGH PORTION
             -   OF THE
                  *   SCREEN. (ROWS 1 THRU 16)
                  *
                  * PARAMETERS:
                  *   - A.- POINTER TO THE FIRST LOCATION OF A STRING OF CONTRO
             -   L CHARACTERS
                  *           USED TO FORMAT THE ROWS ON THE HIGH PORTION OF THE
             -   SCREEN.
                  *
                  ***************************************************************
             -   ***********/
  78   1          INTERP: PROCEDURE (A);
  79   2             DCL A ADDRESS,
                         ARRAY BASED A (8Ø) BYTE,
                         (I,NUM) BYTE;
  8Ø   2             CALL START$PROT$FIELD;
  81   2             I = Ø;
  82   2             DO WHILE ARRAY(I) <> '#';
  83   3                IF ARRAY(I) = '$'
                        THEN DO;
  85   4                   CALL STOP$PROT$FIELD;
  86   4                   I = I + 1;
  87   4                   NUM = ARRAY(I) - 3ØH;
  88   4                   CALL PUT$SPACE(NUM);
  89   4                   CALL START$PROT$FIELD;
  9Ø   4                   END;
  91   3                ELSE DO;
  92   4                   IF ARRAY(I) = '%'
                           THEN DO;
```

```
 94    5                  I = I + 1;
 95    5                  NUM = ARRAY(I) - 30H;
 96    5                  CALL PUT$SPACE(NUM);
 97    5                  END;
 98    4                ELSE DO;
 99    5                  CALL CRT$WRITE(ARRAY(I));
100    5                  END;
101    4                END;
102    3              I = I + 1;
103    3              END;
104    2            CALL STOP$PHOT$FIELD;
105    2            END INTERP;


               /*****************************************************************
           -    ********
               *
               * INIT$HIGH$SCREEN:
               *   THIS PROCEDURE IS USED TO INITIALIZE THE HIGH PORTION OF
           -    THE
               *   SCREEN, WITH A FORMAT PRE-STABLISHED.
               *
               *****************************************************************
           -    ********/
106    1       INIT$HIGH$SCREEN: PROCEDURE PUBLIC;
107    2         CALL STOP$PROT$FIELD;                       /* TO END ROLL MOD
           -    E, IF SET. */
108    2         CALL SET$HIGH$HOME;
109    2         CALL INTERP(.LIN1A);                         /* LINE 1. */
```

```
110   2          CALL INTERP(.LIN2A);                         /* LINE 2. */
111   2          CALL INTERP(.LIN3A);                         /* LINE 3. */
112   2          CALL INTERP(.LIN4A);                         /* LINE 4. */
113   2          CALL INTERP(.LIN5A);                         /* LINE 5. */
114   2          CALL INTERP(.LIN6A);                         /* LINE 6. */
115   2          CALL INTERP(.LIN6A);                         /* LINE 7. */
116   2          CALL INTERP(.LIN6A);                         /* LINE 8. */
117   2          CALL INTERP(.LIN6A);                         /* LINE 9. */
118   2          CALL INTERP(.LIN6A);                        /* LINE 10. */
119   2          CALL INTERP(.LIN6A);                         /* LINE 11. */
120   2          CALL INTERP(.LIN6A);                        /* LINE 12. */
121   2          CALL INTERP(.LIN6A);                        /* LINE 13. */
122   2          CALL INTERP(.LIN6A);                         /* LINE 14. */
123   2          CALL INTERP(.LIN6A);                        /* LINE 15. */
124   2          CALL INTERP(.LIN5A);                         /* LINE 16. */
125   2          END INIT$HIGH$SCREEN;


                 /*****************************************************************
             -    *************
                 *
                 * PRINT$TIME$ZONE:
                 *   THIS PROCEDURE WILL PRINT THE CURRENT TIME ZONE NUMBER.
                 *
                 * PARAMETERS:
                 *   - A.- POINTER TO A STRING CONTAINING THE ASCII CHARACTERS
             -    REPRESENTING
                 *     THE TIME ZONE NUMBER.
                 *
```

```
                  ****************************************************************
               -   ************/
 126   1         PRINT$TIME$ZONE: PROCEDURE (A) PUBLIC;
 127   2            DCL A ADDRESS,
                        BUFFER BASED A (5) BYTE;
 128   2            CALL SET$HIGH$HOME;
 129   2            CALL PUT$TAB(15);
 130   2            CALL CRT$PRINT$STRING(.BUFFER);
 131   2            CALL SET$LOW$HOME;
 132   2            END PRINT$TIME$ZONE;


                 /****************************************************************
               -   **************
                 *
                 * PRINT$TIME:
                 *   THIS PROCEDURE WILL PRINT THE LOCAL TIME AT THE CRT.
                 *
                 * PARAMETERS:
                 *   - A.- POINTER TO A STRING OF ASCII CHARACTERS REPRESENTIN
               -   G THE TIME.
                 *
                 ****************************************************************
               -   **************/
 133   1         PRINT$TIME: PROCEDURE (A) PUBLIC;
 134   2            DCL A ADDRESS,
                        I BYTE,
                        BUFFER BASED A (8) BYTE;
 135   2            CALL SET$HIGH$HOME;
 136   2          CALL PUTSTAB(1);
 137   2          DO I = 0 TO 5;
 138   3             CALL CRT$WRITE(BUFFER(I));
 139   3             END;
 140   2          CALL SET$LOW$HOME;
 141   2          END PRINT$TIME;


               /***************************************************************
            -   ************
               *
               * PRINT$LAT$LONG:
               *   THIS PROCEDURE WILL DISPLAY THE CURRENT LATITUDE AND LONG
            -   ITUDE AT THE
               *   CRT.
               *
               * PARAMETERS:
               *   - A.- POINTER TO A STRING OF ASCII CHARACTERS REPRESENTIN
            -   G THE LATITUDE.
               *   - B.- POINTER TO A STRING OF ASCII CHARACTERS REPRESENTIN
            -   G THE LONGITUDE.
               *
               ****************************************************************
            -   **************/
 142   1       PRINT$LAT$LONG: PROCEDURE (A,B) PUBLIC;
 143   2          DCL (A,B)ADDRESS,
                      BUFF1 BASED A (10) BYTE,
                      BUFF2 BASED B (10) BYTE;
 144   2          CALL SET$HIGH$HOME;
```

```
145   2          CALL PUT$TAB(4);
146   2          CALL CRT$PRINT$STRING(.BUFF1);
147   2          CALL SET$HIGH$HOME;
148   2          CALL PUT$TAB(16);
149   2          CALL CRT$PRINT$STRING(.BUFF2);
150   2          CALL SET$LOW$HOME;
151   2          END PRINT$LAT$LONG;


                /**************************************************************
            -    **************
                *
                * PRINT$COURSE:
                *   THIS PROCEDURE WILL PRINT THE CURRENT OWN COURSE VALUE AT
            -    THE
                *   CRT.
                *
                * PARAMETERS:
                *   - A.- POINTER TO A STRING OF ASCII CHARACTERS REPRESENTIN
            -    G THE COURSE.
                *
                **************************************************************
            -    *************/
152   1     PRINT$COURSE: PROCEDURE (A) PUBLIC;
153   2          DCL A ADDRESS,
                     BUFFER BASED A (6) BYTE;
154   2          CALL SET$HIGH$HOME;
155   2          CALL PUT$CR(1);
156   2          CALL PUT$TAB(2);
```

```
 157    2          CALL CRT$PRINT$STRING(.BUFFER);
 158    2          CALL SET$LOW$HOME;
 159    2          END PRINT$COURSE;


                /****************************************************************
          -      *************
                *
                * PRINT$SPEED:
                *   THIS PROCEDURE WILL PRINT THE CURRENT OWN SPEED AT THE CR
          -      T.
                *
                * PARAMETERS:
                *   - A.- POINTER TO A STRING OF ASCII CHARACTERS REPRESENTIN
          -      G THE SPEED.
                *
                *****************************************************************
          -      *************/
 160    1       PRINT$SPEED: PROCEDURE (A) PUBLIC;
 161    2          DCL A ADDRESS,
                       BUFFER BASED A (5) BYTE;
 162    2          CALL SET$HIGH$HOME;
 163    2          CALL PUT$CR(1);
 164    2          CALL PUT$TAB(5);
 165    2          CALL CRT$PRINT$STRING(.BUFFER);
 166    2          CALL SET$LOW$HOME;
 167    2          END PRINT$SPEED;
```

```
                 /***********************************************************
               -  ***********
                  *
                  * PRINT$CONTACTS:
                  *   THIS PROCEDURE WILL PRINT THE NUMBER OF FRIEND, HOSTILE A
               -  ND UNKNOWN
                  *   CONTACTS BEING PROCESSED BY THE SYSTEM.
                  *
                  * PARAMETERS:
                  *   - A.- POINTER TO A STRING OF ASCII CHARACTERS REPRESENTIN
               -  G THE NUMBER
                  *     OF FRIEND, HOSTILE AND UNKNOWN CONTACTS.
                  *
                  ************************************************************
               -  *************/
 168   1          PRINT$CONTACTS: PROCEDURE (A) PUBLIC;
 169   2             DCL A ADDRESS,
                         BUFFER BASED A (8) BYTE;
 170   2             CALL SET$HIGH$HOME;
 171   2             CALL PUT$CR(1);
 172   2             CALL PUT$TAB(8);
 173   2             CALL CRT$PRINT$STRING(.BUFFER);
 174   2             CALL SET$LOW$HOME;
 175   2             END PRINT$CONTACTS;


                  /**********************************************************
               -   **********
```

```
                 *
                 * PRINT$MODE:
                 *   THIS PROCEDURE WILL PRINT THE CURRENT OPERATING MODE.
                 *
                 * PARAMETERS:
                 *   - A.- POINTER TO A STRING OF ASCII CHARACTERS DEFINING TH
               -  E CURRENT
                 *     OPERATING MODE.
                 *
                 ****************************************************************
               -  ***********/
 176   1         PRINT$MODE: PROCEDURE (A) PUBLIC;
 177   2            DCL A ADDRESS,
                        BUFFER BASED A (9) BYTE;
 178   2            CALL SET$HIGH$HOME;
 179   2            CALL PUT$CR(1);
 180   2            CALL PUT$TAB(11);
 181   2            CALL CRT$PRINT$STRING(.BUFFER);
 182   2            CALL SET$LOW$HOME;
 183   2            END PRINT$MODE;


                 /***************************************************************
               -  ***********
                 *
                 * PRINT$CONTACT$INFO:
                 *   THIS PROCEDURE WILL PRINT ALL THE CURRENT INFORMATION OF
               -  ANY CONTACT
                 *   BEING DISPLAYED.
```

```
                  *
                  * PARAMETERS:
                  *   - NUM.- REPRESENTS THE RELATIVE POSITION FROM TOP TO BOTT
                -  OM, OF THE
                  *     CONTACT LINE DESIRED TO BE UPDATED.
                  *   - A.- POINTER TO A STRING OF ASCII CHARACTERS REPRESENTIN
                -  G THE CONTACT
                  *     INFORMATION TO BE DISPLAYED.
                  *
                  *************************************************************
                -  *************/
  184   1         PRINT$CONTACT$INFO: PROCEDURE (NUM,A) PUBLIC;
  185   2            DCL A ADDRESS,
                         BUFFER BASED A (44) BYTE,
                         NUM BYTE;
  186   2            CALL SET$HIGH$HOME;
  187   2            CALL PUT$CR(4);
  188   2            CALL PUT$CR(NUM);
  189   2            CALL CRT$PRINT$STRING(.BUFFER);
  190   2            CALL SET$LOW$HOME;
  191   2            END PRINT$CONTACT$INFO;



  192   1         END CRT;


MODULE INFORMATION:
```

```
PL/M-80 COMPILER                                                22 MAR 81  PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE FLTASCII
OBJECT MODULE PLACED IN FPCONV.OBJ
COMPILER INVOKED BY:  :F1:PLM80 FPCONV.SRC PAGELENGTH(33) PAGEWIDTH(75) DAT
                      -E(22 MAR 81)



   1          FLTASCII: DO;

                                                  /***   EXTERNALS:   ***/


   2    1     EDIV:
                PROCEDURE (A,B,C,D) EXTERNAL;
   3    2       DECLARE (A,B,C,D) ADDRESS; END;

   5    1     FMUL:
                PROCEDURE (A,B,C) EXTERNAL;
   6    2       DECLARE (A,B,C) ADDRESS; END;

   8    1     FDIV:
                PROCEDURE (A,B,C) EXTERNAL;
   9    2       DECLARE (A,B,C) ADDRESS; END;

  11    1     FADD:
                PROCEDURE (A,B,C) EXTERNAL;
  12    2       DECLARE (A,B,C) ADDRESS; END;
```

```
 14    1       FSUB:
                  PROCEDURE (A,B,C) EXTERNAL;
 15    2          DECLARE (A,B,C) ADDRESS; END;

 17    1       FLTDS:
                  PROCEDURE (A,B) EXTERNAL;
 18    2          DECLARE (A,B) ADDRESS; END;

 20    1       FIXSD:
                  PROCEDURE (A,B) EXTERNAL;
 21    2          DECLARE (A,B) ADDRESS; END;

 23    1       FZTST:
                  PROCEDURE (A,B) BYTE EXTERNAL;
 24    2          DECLARE (A,B) ADDRESS; END;

 26    1        DECLARE LIT LITERALLY 'LITERALLY',
                        DCL LIT 'DECLARE';

                /**************************************************************
                 ****************
                *
                * ASCIISTOSFLOAT:
                *   PROCEDURE USED TO CONVERT A STRING OF ASCII CHARACTERS IN
              -  TO A FLOATING
                *   POINT NUMBER, ACCORDING TO THE FORMAT REQUIRED TO OPERATE
              -   ON THE F.P.
                *   BOARD.
```

```
*
* PARAMETERS:
*    - A.- POINTER TO A N BYTE VECTOR CONTAINING THE ASCII STR
-  ING OF NUMBERS
*       REPRESENTING A DECIMAL VALUE DESIRED TO BE REPRESENTED
-  IN FLOATING
*       POINT FORMAT. IF THE N BYTE VECTOR IS REPRESENTED BY TH
-  E
*       NAME 'BUFFER', THEN THE FOLLOWING RULES APPLY:
*          BUFFER(0).- CONTAINS THE DECIMAL NUMBER OF ASCII CH
-  ARACTERS PRE-
*                      SENT IN THE BUFFER. A VALUE OF 0 WILL D
-  ENOTE THE NUM-
*                      BER 0.0. .
*          BUFFER(1).- CONTAINS THE DECIMAL NUMBER OF ASCII CH
-  ARACTERS THAT
*                      REPRESENT THE INTEGER PORTION OF THE NU
-  MBER. A VALUE OF
*                      0 WILL MEAN THAT THE NUMBER IS LESS THA
-  N ONE.
*          BUFFER(2) --> BUFFER(N-2).- EACH BYTE CONTAINS AN A
-  SCII CHA-
*                      RACTER. (HEXADECIMAL VALUE)
*          BUFFER(N-1).- A VALUE OF DECIMAL 1 WILL MEAN THAT T
-  HE NUMBER IS NE-
*                      GATIVE.
*
*
*    - LIMIT.- TOTAL NUMBER OF BYTES IN 'BUFFER'. (N)
*
```

```
                  *
                  *   - B.- POINTER TO A FOUR BYTE VECTOR THAT WILL CONTAIN THE
                -   FLOATING POINT
                  *          REPRESENTATION OF THE ASCII STRING CONTAINED IN BUF
                -   FER(2) THRU
                  *          BUFFER(N).
                  *
                  ****************************************************************
                -   ******************/
  27   1          ASCII$TO$FLOAT:PROCEDURE(A,LIMIT,B) PUBLIC;
  28   2             DCL (A,B) ADDRESS,
                        BUFFER BASED A BYTE,
                        VECTOR BASED B (4) BYTE,
                        TEN$FLOAT (4) BYTE DATA (00H,00H,20H,41H),
                        TEMP (4) BYTE,
                        RESULT (4) BYTE,
                        (I,NUMINT,NUMDEC,NUM,LIMIT,T0,T1) BYTE;

  29   2             T0 = BUFFER;
  30   2             A = A + 1;
  31   2             T1 = BUFFER;
  32   2             DO I = 0 TO 3;              /* INITIALIZE VECTOR. */
  33   3                VECTOR(I) = 0;
  34   3                END;
                  /*   CHECK IF PROPOSED NUMBER IS 0.   */
  35   2             IF T0 = 0 THEN RETURN;
  37   2             NUMINT = T1;
  38   2             NUMDEC = T0 - T1;
                  /*   FIND INTEGER PORTION OF NUMBER.   */
  39   2             DO WHILE NUMINT > 0;
  40   3          DO I = 0 TO 3;
  41   4             TEMP(I) = 0;
  42   4             END;
  43   3          A = A + 1;
  44   3          TEMP(0) = BUFFER - 30H;
  45   3          CALL FLTDS(.TEMP,.RESULT);
  46   3          IF NUMINT > 1 THEN DO;
  47   4             DO I = 1 TO NUMINT - 1;
  49   5                CALL FMUL(.RESULT,.TENSFLOAT,.RESULT);
  50   5                END;
  51   4             END;
  52   3          CALL FADD(.RESULT,.VECTOR,.VECTOR);
  53   3          NUMINT = NUMINT - 1;
  54   3          END;

        /*   FIND DECIMAL PORTION OF NUMBER PROPOSED.   */
  55   2       NUM = 0;
  56   2       DO WHILE NUMDEC > 0;
  57   3          DO I = 0 TO 3;
  58   4                TEMP(I) = 0;
  59   4                END;
  60   3          A = A + 1;
  61   3          NUM = NUM + 1;
  62   3          TEMP(0) = BUFFER - 30H;
  63   3          CALL FLTDS(.TEMP,.RESULT);
  64   3          DO I = 1 TO NUM;
  65   4             CALL FDIV(.RESULT,.TENSFLOAT,.RESULT);
  66   4             END;
  67   3          CALL FADD(.RESULT,.VECTOR,.VECTOR);
  68   3          NUMDEC = NUMDEC - 1;
```

```
 69   3            END;

               /*   CHECK FOR SIGN OF NUMBER PROPOSED.   */
 70   2          A = A + 1;
 71   2          IF BUFFER = 1 THEN VECTOR(3) = VECTOR(3) XOR 80H;

               /*   ALL DONE. RETURN TO CALLING MODULE. */
 73   2          RETURN;
 74   2          END ASCIISTOSFLOAT;

               /*************************************************************
             -  ******************
               *
               * FRAC$TO$ASCII:
               *   PROCEDURE USED TO CONVERT A FRACTIONAL PART OF A F.P. NUM
             -  BER
               *   INTO AN ASCII REPRESENTATION AND STORE IT IN A BUFFER.
               *
               * PARAMETERS:
               *   - A -POINTER TO A VARIABLE - WHICH CONTAINS THE FRACTIONA
             -  L PART OF A
               *          F.P. NUMBER- PASSED BY FLOAT$TO$ASCII PROCEDURE.
               *   - B -POINTER TO A BUFFER PARTIALLY FILLED W/ THE INTEGER
             -  PART OF
               *          A F.P. NUMBER CONVERTED TO ASCII BY FLOAT$TO$ASCII.
               *   - DEC$POS -POINTER TO A VARIABLE WHICH INDICATES THE POSI
             -  TION OF DECI-
               *            MAL POINT TO BE SET ; IT RETURN THE ADDRESS WH
             -  ICH CONTAINS
```

```
                 *                 THE FIRST VACANT POSITION IN THE BUFFER AFTER
           -     FILLED.
                 *
                 ******************************************************************
           -     ****************/
 75    1         FRAC$TO$ASCII: PROCEDURE (A,B,DEC$POS);
 76    2            DCL (A,B,DEC$POS) ADDRESS;
 77    2            DCL TERM BASED A (4) BYTE,
                        BUFF BASED B (26) BYTE,
                        C BASED DEC$POS BYTE;
 78    2            DCL TEN$FLOAT (4) BYTE DATA (00H,00H,20H,41H);
 79    2            DCL TEMP (4) BYTE, TEMP1 BYTE DATA (4),
                        (FLAG, I) BYTE;

 80    2            BUFF(C) = '.';
 81    2            C = C + 1;
 82    2            FLAG = 0FFH;
 83    2            DO WHILE FLAG;            /* SET UP FRAC PART OF F.P. NUMBER
           -       TO ASCII */
 84    3               CALL FMUL (.TERM, .TEN$FLOAT, .TERM);
 85    3               CALL FIXSD (.TERM,.TEMP);
 86    3               BUFF(C) = TEMP(0) + 30H;
 87    3               CALL FLTIS (.TEMP, .TEMP);
 88    3               CALL FSUB (.TERM, .TEMP, .TERM);
 89    3               C = C + 1;
 90    3               IF C = 26 THEN RETURN;
 92    3               FLAG = NOT (FLAG:= FZTST (.TERM, .TEMP1));
 93    3               END;
 94    2            RETURN;
 95    2            END FRAC$TO$ASCII;
```

```
             /************************************************************
         -    *******************
              *
              * FLOAT$TO$ASCII:
              *    PROCEDURE USED TO CONVERT A FLOATING POINT NUMBER -SEE I.
         -    TEL SEC 310-
              *    INTO AN ASCII REPRESENTATION AND STORE IT IN A BUFFER TO
         -    BE SENT TO THE
              *    CALLING MODULE.
              *
              * PARAMETERS:
              *     - FLOAT -POINTER TO A VARIABLE -WHICH CONTAINS THE F.P. N
         -    UMBER- PASSED
              *               BY THE CALLING MODULE;
              *     - ASC$BUFFER -POINTER TO A BUFFER W/ LENGTH 28 WHICH CONT
         -    AINS THE ASCII
              *                       REPRESENTATION OF THE F.P. NO.;
              *     - END$BUFFER - POINTER TO VARIABLE PASSED TO THE CALLING
         -    MODULE INDICA-
              *                        TING THE FIRST VACANT BYTE OF BUFFER AFTER
         -     LAST PRINTABLE
              *                        ASCII BYTE.
              *
              ************************************************************
         -     *******************/
 95   1      FLOAT$TO$ASCII: PROCEDURE (FLOAT,ASC$BUFFER,END$BUFFER) PUBLI
         -    C;
 97   2        DCL (FLOAT,ASC$BUFFER,END$BUFFER) ADDRESS,
                              M BASED FLOAT (4) BYTE,
                              A BASED ASCSBUFFER (28) BYTE,
                              DECSPOINT BASED FNISBUFFER BYTE,
                              ONE (4) BYTE DATA (22H,00H,80H,3FH),
                              TEN (2) BYTE DATA (0AH,00F),
                              (TEMP,TEMP2,FRAC,REM) (4) BYTE,
                              (TEMP1,I,SIGN,EXP,FLAG) BYTE;

  98   2         DO I = Ø TO LAST(A);                          /* INITIALIZE BUFFER */
  99   3             A(I) = ' ';
 1ØØ   3             END;
 1Ø1   2         SIGN = M(3) AND 8ØH;
 1Ø2   2         IF SIGN = 8ØH THEN A(Ø) = '-';
                                ELSE A(Ø) = ' ';
 1Ø4   2         EXP = SHL(M(3),1) OR SHR(M(2),7);
 1Ø5   2         TEMP(3) = M(3); TEMP(2) = M(2); TEMP(1) = M(1); TEMP(Ø) =
          -     M(Ø);
 11Ø   2         IF ((EXP < 7FE) AND (EXP >= ØØH)) THEN I = Ø;
 112   2         IF EXP = 7FH THEN I = 1;
 114   2         IF ((EXP <= ØFEH) AND (EXP > 7FH)) THEN I = 2;
 116   2         DO CASE I;
 117   3           DO;                 /* CASE Ø: APPLIED FOR -1.Ø < F.P. NUM
          -     BERS < 1.0 */
 118   4           DECSPOINT = 2;
 119   4           A(1) = 'Ø';
 12Ø   4           TEMP(3) = TEMP(3) AND 7FH;
 121   4           CALL FRACSTO$ASCII (.TEMP,.A,.DECSPOINT);
 122   4           END;

 123   3           DO;
```

```
                /* CASE 1: APPLIED FOR 2.Ø > F.P. NUMBER >= 1.Ø
                        OR -2.0 < F.P. NUMBER <= -1.Ø */
124    4            DEC$POINT = 2;
125    4            A(1) = '1';
126    4            TEMP(3) = TEMP(3) AND 7FH;
127    4            CALL FSUB (.TEMP,.ONE,.TEMP);
128    4            CALL FRAC$TO$ASCII (.TEMP,.A,.DEC$POINT);
129    4            END;

130    3            DO;
                /* CASE 2: APPLIED FOR -1.Ø > F.P. NUMBER > 1.Ø */
131    4            DEC$POINT = 1;
132    4              EXP = EXP - 7FH;           /* TAKE OUT BIAS OF EXPONENT
         -        */
133    4              DO I = 1 TO 23 - EXP;
                /* NEXT 2 INTERACTIVES "DO" TAKE OUT FRACTIONAL PART OF F.P.
         -       NUMBER */
134    5                   TEMP(2) = SCR(TEMP(2),1);
135    5                   TEMP(1) = SCR(TEMP(1),1);
136    5                   IF CARRY THEN TEMP(Ø) = SHR(TEMP(Ø),1) OR 8ØH;
138    5                             ELSE TEMP(Ø) = SHR(TEMP(Ø),1);
139    5                 END;
140    4            EXP = EXP OR EXP;             /* RESET CARRY BIT */
141    4            DO I = 1 TO 23 -EXP;
142    5                 TEMP(Ø) = SCL(TEMP(Ø),1);
143    5                 TEMP(1) = SCL(TEMP(1),1);
144    5                 IF CARRY THEN TEMP(2) = SHL(TEMP(2),1) OR Ø1H;
146    5                           ELSE TEMP(2) = SHL(TEMP(2),1);
147    5                 END;
148    4            CALL FSUB (.M,.TEMP,.FRAC);            /* SAVE FRAC PART
```

```
              -   F F.P. NUMBER */
 149    4             FRAC(3) = FRAC(3) AND 7FH;
 150    4             CALL FIXSD (.TEMP,.TEMP);
 151    4             IF (TEMP1 := TEMP(3) AND 80H) = 80H THEN
 152    4                 DO;          /* PERFORMS TWO'S COMPLEMENT OF A NEGA
              -   TIVE F.P. NUMBER */
 153    5                 TEMP(0) = NOT TEMP(0); TEMP(1) = NOT TEMP(1);
 155    5                 TEMP(2) = NOT TEMP(2); TEMP(3) = NOT TEMP(3);
 157    5                 TEMP(0) = TEMP(0) + 1; TEMP(1) = TEMP(1) PLUS 0;
 159    5                 TEMP(2) = TEMP(2) PLUS 0; TEMP(3) = TEMP(3) PLUS 0;
 161    5                 END;
 162    4             FLAG = 0FFH;
 163    4             DO WHILE FLAG;           /* SET UP INTEGER PART OF F.P. N
              -   UMBER TO ASCII */
 164    5                CALL EDIV (.TEMP,.TEN,.TEMP,.REM);
 165    5                A(DEC$POINT) = REM(0) + 30H;
 166    5                DEC$POINT = DEC$POINT + 1;
 167    5                CALL FLTDS (.TEMP,.TEMP);
 168    5                TEMP1 = 4;
 169    5                FLAG = NOT (FLAG:= FZTST (.TEMP, .TEMP1));
 170    5                CALL FIXSD (.TEMP,.TEMP);
 171    5                END;
 172    4             DO I = 1 TO DEC$POINT/2;
 173    5                TEMP1 = A(I);
 174    5                A(I) = A(DEC$POINT - I);
 175    5                A(DEC$POINT - I) = TEMP1;
 176    5                END;
 177    4             CALL FRAC$TO$ASCII (.FRAC,.A,.DEC$POINT);
 178    4             END;
 179    3          END;
 180    2         RETURN;
 181    2         END FLOATSTO$ASCII;

 182    1       END FITASCII;


MODULE INFORMATION:

     CODE AREA SIZE     = 0511H   1297D
     VARIABLE AREA SIZE = 003AH     58D
     MAXIMUM STACK SIZE = 0006H      6D
     257 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER                                        20 MAR 81  PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE FLOATINGPOINT
OBJECT MODULE PLACED IN FLOAT.OBJ
COMPILER INVOKED BY:  :F1:PLM80 FLOAT.SRC PAGELENGTH(33) PAGEWIDTH(75) DATE
                     -(20 MAR 81)



   1          FLOATING$POINT: DO;

                                        /***   DECLARATIONS:   ***/

   2    1     DECLARE LIT LITERALLY 'LITERALLY',
                    DCL LIT 'DECLARE';
   3    1     DCL
                 MEBAS ADDRESS PUBLIC DATA (0F790H) ,
                 RES$TABLE (8) BYTE PUBLIC AT (0F790H), /* MEMORY RESERVED
      -         FOR F.P. BOARD. */
                 OUT$OP$CODE LIT '010H',  /* BASE OUTPUT PORT FOR F.P. BOA
      -        RD. */
                 IN$STATUS LIT '011H',    /* INPUT PORT FOR STATUS OF F.P.
      -         BOARD. */
                 IN$FLAG LIT '017H',      /* INPUT PORT FOR FLAG FROM F.P.
      -         BOARD. */
                 MEM$LOW LIT '011H',      /* OUTPUT PORT FOR LOW PORTION O
      -        F MEMORY BASE.*/
                 MEM$HIGH LIT '012H',     /* OUTPUT PORT FOR HIGH PORTION
      -        OF MEMORY BASE. */
                 MUL$CODE LIT '00H',      /* FIXED POINT MULTIPLICATION CO
      -   DE. */
             DIV$CODE LIT '01H',        /* FIXED POINT DIVISION CODE. */
             FMUL$CODE LIT '02H',       /* F.P. MULTIPLICATION CODE. */
             FDIV$CODE LIT '03H',       /* F.P. DIVISION CODE. */
             FADD$CODE LIT '04H',       /* F.P. ADD CODE. */
             FSUB$CODE LIT '05H',       /* F.P. SUBTRACT CODE. */
             FSQR$CODE LIT '06H',       /* F.P. SQUARE CODE. */
             FSQRT$CODE LIT '07H',      /* F.P. SQUARE ROOT CODE. */
             FLTDS$CODE LIT '08H',      /* FIXED-TO-FLOAT CONVERSION COD
      -   E. */
             FIXSD$CODE LIT '09H',      /* FLOAT-TO-FIXED CONVERSION COD
      -   E. */
             FCMPR$CODE LIT '0AH',      /* F.P. COMPARE CODE. */
             FZTST$CODE LIT '0BH',      /* F.P. TEST CODE. */
             EXCH$CODE LIT '0FH',       /* EXCHANGE CODE. */
             EDIV$CODE LIT '0EH',       /* EXTENDED FIXED POINT DIVISION
      -    CODE. */
             BUSY$MASK LIT '01H',       /* MASK FOR BUSY SIGNAL FROM F.P
      -   . BOARD. */
             ERROR$MASK LIT '04H';      /* MASK FOR ERROR CODE FROM F.P.
      -    BOARD. */


                                   /***   EXTERNALS:   ***/


  4  1      CRT$PRINT$STRING:PROCEDURE (A) EXTERNAL;
  5  2        DECLARE A ADDRESS;
  6  2        END CRT$PRINT$STRING;
  7   1       SENDSCRLF: PROCEDURE EXTERNAL;
  8   2          END SENDSCRLF;


              /*************************************************************
            -  *******
              *
              * INITSFP:
              *   PROCEDURE USED TO INITIALIZE THE FLOATING POINT MODULE.
              *
              * USAGE:
              *   THIS PROCEDURE SHOULD BE CALLED ONE TIME FROM THE USER'S
              *   MAIN PROGRAM BEFORE ATTEMPTING TO USE ANY OF THE ROUTINES
            -   PROVI-
              *   DED FOR FLOATING POINT OPERATIONS WITH THE FLOATING POINT
            -   BOARD.
              *
              **************************************************************
            -  *******/
  9   1       INITSFP: PROCEDURE PUBLIC;
 10   2          OUTPUT(MEM$LOW) = LOW(MEBAS);
 11   2          OUTPUT(MEM$HIGH) = HIGH(MEBAS);
 12   2          RETURN;
 13   2          END INITSFP;


              /*************************************************************
            -  *****
```

```
                  *
                  * ADJUST$OP:
                  *   PROCEDURE USED TO PUT TWO VECTORS INTO THE TABLE VECTOR.
                  *
                  * PARAMETERS:
                  *   - A,B.- POINTERS TO FIRST AND SECOND VECTOR VALUES.
                  *
                  ******************************************************************
                   ****/
  14    1         ADJUST$OP: PROCEDURE(A,B);
  15    2            DCL (A,B) ADDRESS,
                         OP1 BASED A (4) BYTE,
                         OP2 BASED B (4) BYTE,
                          I BYTE;
  16    2            DO I = 0 TO LAST(OP1);
  17    3               RES$TABLE(I) = OP1(I);
  18    3               RES$TABLE(I + 4) = OP2(I);
  19    3               END;
  20    2            RETURN;
  21    2            END ADJUST$OP;

                  /******************************************************************
                   ****
                  *
                  * ADJUST1$OP:
                  *   PROCEDURE USED TO PUT ONE VECTOR INTO THE TABLE VECTOR.
                  *
                  * PARAMETERS:
                  *   -  A.- POINTER TO VECTOR VALUE.
```

```
                   *
                   ****************************************************************
               -    ***/
 22   1            ADJUST1$OP: PROCEDURE (A);
 23   2               DCL A ADDRESS,
                          OP1 BASED A (4) BYTE,
                          I BYTE;
 24   2               DO I = 0 TO LAST(OP1);
 25   3                  RES$TABLE(I) = OP1(I);
 26   3                  END;
 27   2               RETURN;
 28   2               END ADJUST1$OP;


                    /***************************************************************
               -     ******
                    *
                    * ADJUST2$OP:
                    *   PROCEDURE USED TO PUT TWO ADDRESS VALUES INTO THE TABLE V
               -     ECTOR.
                    *
                    * PARAMETERS:
                    *    - A,B.- POINTERS TO TWO ADDRESS VALUES.
                    *
                    ****************************************************************
               -     *****/
 29   1             ADJUST2$OP: PROCEDURE (A,B);
 30   2                DCL (A,B) ADDRESS,
                           OP1 BASED A (2) BYTE,
                           OP2 BASED B (2) BYTE,
                      I BYTE;
  31   2          DO I = 0 TO LAST(OP1);
  32   3             RES$TABLE(I) = OP1(I);
  33   3             RES$TABLE(I + 4) = OP2(I);
  34   3             END;
  35   2          RETURN;
  36   2          END ADJUST2$OP;


               /*****************************************************************
           -    **
               *
               * VAL$RESULT:
               *   PROCEDURE USED TO GET THE RESULT IN FIRST FOUR BYTES OF T
           -    HE
               *   TABLE VECTOR, AND PUT IT INTO ANOTHER VECTOR PROVIDED.
               *
               * PARAMETERS:
               *   - C.- POINTER TO VECTOR ON WHICH RESULT IS DESIRED.
               *
               *****************************************************************
           -    **/
  37   1       VAL$RESULT: PROCEDURE(C);
  38   2          DCL C ADDRESS,
                      RESULT BASED C (4) BYTE,
                      I BYTE;
  39   2          DO I = 0 TO LAST(RESULT);
  40   3             RESULT(I) = RES$TABLE(I);
  41   3             END;
  42   2          RETURN;
```

```
43   2          END VALSRESULT;

               /*******************************************************************
        -       **
               *
               * VAL$RESULT$1:
               *   PROCEDURE USED TO PUT ALL EIGHT BYTES OF THE TABLE VECTOR
               *   INTO TWO FOUR BYTES VECTORS PROVIDED.
               *
               * PARAMETERS:
               *    - A,B.- POINTERS TO VECTORS ON WHICH RESULT IS DESIRED. A
               *       MUST POINT TO FIRST VECTOR (FIRST FOUR BYTES OF
               *       TABLE), AND B MUST POINT TO THE SECOND VECTOR (LAST FOU
        -       R
               *       BYTES IN TABLE).
               *
               *******************************************************************
        -       */
44   1         VAL$RESULT$1: PROCEDURE(A,B);
45   2            DCL (A,B) ADDRESS,
                      OP1 BASED B (4) BYTE,
                      OP2 BASED A (4) BYTE,
                      I BYTE;
46   2            DO I = 0 TO LAST(OP1);
47   3               OP2(I) = RES$TABLE(I);
48   3               OP1(I) = RES$TABLE(I + 4);
49   3               END;
50   2            RETURN;
51   2            END VAL$RESULT$1;
```

```
                /*****************************************************************
              -  **
                 *
                 * VAL$RESULT$2:
                 *   PROCEDURE USED TO GET THE RESULT FROM FIXED POINT
                 *   DIVISION OPERATION, AND PUT THEM INTO TWO ADDRESS
                 *   LOCATIONS PROVIDED.
                 *
                 * PARAMETERS:
                 *   - C,R.- POINTERS TO TWO ADDRESS LOCATIONS IN WHICH THE
                 *      RESULT IS DESIRED TO BE PLACED.
                 *
                 ******************************************************************
              -  **/
  52   1        VAL$RESULT$2: PROCEDURE (C,R);
  53   2          DCL (C,R) ADDRESS,
                       OP1 BASED C (2) BYTE,
                       OP2 BASED R (2) BYTE,
                       I BYTE;
  54   2          DO I = 0 TO LAST(OP1);
  55   3            OP1(I) = RES$TABLE(I);
  56   3            OP2(I) = RES$TABLE(I + 4);
  57   3            END;
  58   2          END VAL$RESULT$2;

                /*****************************************************************
```

```
               -  *******
                  *
                  * COMPARE:
                  *   PROCEDURE USED TO CHECK FOR OUTPUT CONDITIONS FROM F.P. B
               -  OARD.
                  *
                  * PARAMETERS:
                  *   - A.- BYTE VALUE CONTAINING RESULT FROM F.P. BOARD.
                  *   - B.- BYTE VALUE CONTAINING CONDITION DESIRED TO BE CHECK
               -  ED:
                  *             <  ..... Ø
                  *             <= ..... 1
                  *             >  ..... 2
                  *             >= ..... 3
                  *             =  ..... 4
                  *             <> ..... 5
                  * USAGE:
                  *   TYPED PROCEDURE THAT RETURNS A 'TRUE' VALUE (ØØ1H) IF OUT
               -  PUT
                  *   CONDITION FROM F.P. BOARD AND CONDITION DESIRED TO BE TES
               -  TED
                  *   ARE SIMILAR. IF NOT SIMILAR, A 'FALSE' VALUE (ØØØH) IS RE
               -  TURNED.
                  *
                  ***************************************************************
               -  ******/
  59   1          COMPARE: PROCEDURE (A,B) BYTE;
  6Ø   2            DCL TRUE LIT 'Ø1H',
                        FALSE LIT 'ØØH';
  61   2            DCL (A,B) BYTE,
                         (LESS$THAN,LESS$OR$EQUAL,GREATER$THAN,GREAT$OR$EQUAL,E
                -   QUAL,NOT$EQUAL)
                         BYTE DATA (0,1,2,3,4,5);
  62    2              IF ((A = 80H) AND ((B = LESS$OR$EQUAL) OR (B = GREAT$OR$EQ
                -   UAL) OR
                         (B = EQUAL))) THEN RETURN TRUE;
  64    2              IF ((A = 40H) AND ((B = GREATER$THAN) OR (B = GREAT$OR$EQU
                -   AL) OR
                         (B = NOT$EQUAL))) THEN RETURN TRUE;
  66    2              IF ((A = 20H) AND ((B = LESS$THAN) OR (B = LESS$OR$EQUAL)
                -   OR
                         (B = NOT$EQUAL))) THEN RETURN TRUE;
  68    2              RETURN FALSE;
  69    2              END COMPARE;


                    /*********************************************************************
                -    ****
                    *
                    * FLOAT$MSG$ERROR:
                    *    PROCEDURE USED TO SEND A MESSAGE ERROR ACCORDING TO ERROR
                    *    CODE PROVIDED BY F.P. BOARD.
                    *
                    * USAGE:
                    *    UNTYPED PROCEDURE. IF CONDITION OF ERROR IS DETECTED, THI
                -    S
                    *    PROCEDURE MUST BE CALLED IN ORDER TO OBTAIN ERROR CODE AN
                -    D
                    *    DISPLAY AN APPROPRIATE MESSAGE. NOTE THAT PROGRAM EXECUTI
```

```
PL/M-80 COMPILER                                           20 MAR 81  PAGE  11

                -   ON
                *   WILL BE STOPPED AND INTERRUPTS WILL BE ENABLED IF AN ERRO
                -   R
                *   IS DETECTED.
                *
                ***********************************************************************
                -   ***/
 70    1        FLOAT$MSG$ERROR: PROCEDURE;
 71    2           DCI MSG1(*) BYTE DATA ('DIVISION BY ZERO.$$'),
                       MSG2(*) BYTE DATA ('DOMAIN ERROR.$$'),
                       MSG3(*) BYTE DATA ('OVERFLOW.$$'),
                       MSG4(*) BYTE DATA ('UNDERFLOW.$$'),
                       MSG5(*) BYTE DATA ('INVALID FORMAT FOR FIRST ARGUMENT.
                -   $$'),
                       MSG6(*) BYTE DATA ('INVALID FORMAT FOR SECOND ARGUMENT
                -   .$$'),
                       ERROR(*) BYTE DATA (' FATAL ERROR.$$'),
                       I BYTE;
 72    2           I = INPUT(IN$STATUS) AND 07H;
 73    2           DO CASE I;
 74    3              ;
 75    3              CALL CRT$PRINT$STRING(.MSG1);
 76    3              CALL CRT$PRINT$STRING(.MSG2);
 77    3              CALL CRT$PRINT$STRING(.MSG3);
 78    3              CALL CRT$PRINT$STRING(.MSG4);
 79    3              CALL CRT$PRINT$STRING(.MSG5);
 80    3              CALL CRT$PRINT$STRING(.MSG6);
 81    3              ;
 82    3              END;
 83    2           CALL CRT$PRINT$STRING(.ERROR);
 84   2          CALL SENDSCRLF;
 85   2          RETURN;
 86   2          END FLOATSMSGSERROR;

                /****************************************************************
           -     *************
                *
                *
                * CHECK:
                *    PROCEDURE USED TO CHECK FOR STATUS OF F.P. BOARD AND TO D
           -     ETECT IF
                *    ANY ERROR HAS OCCURRED.
                *
                * USAGE:
                *    UNTYPED PROCEDURE THAT IS CALLED BY ALL PROCEDURES THAT T
           -     RY TO EXECUTE
                *    A FLOATING POINT OPERATION WITH THE F.P. BOARD.
                *
                *****************************************************************
           -     *************/
 87   1         CHECK: PROCEDURE;
 88   2            DCL I BYTE;
 89   2            DO WHILE ((I:=INPUT(INSFLAG)) AND BUSY$MASK) = BUSY$MASK;
 90   3               END;
 91   2            IF (I AND ERROR$MASK) = ERROR$MASK
                       THEN DO;
 93   3                  CALL FLOAT$MSG$ERROR;
 94   3                  HALT;
 95   3                  END;
 96   2          RETURN;
 97   2          END CHECK;

              /**************************************************************
       -       *************
              *
              * MUL:
              *   PROCEDURE USED TO PERFORM FIXED POINT MULTIPLICATION USIN
       -       G THE F.P.
              *   BOARD.
              *
              * PARAMETERS:
              *   - A.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH
       -        THE FIRST
              *          OPERAND WILL BE LOCATED.
              *   - B.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH
       -        THE SECOND
              *          OPERAND WILL BE LOCATED.
              *   - C.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH
       -        THE RESULT
              *          IS DESIRED TO BE PLACED. NOTICE THAT IT COULD BE TH
       -       E SAME
              *          LOCATION USED FOR ANY OF THE OPERANDS.
              *
              ***************************************************************
       -       ***********/
 98   1       MUL: PROCEDURE (A,B,C) PUBLIC;
 99   2          DCL (A,B,C) ADDRESS;
100   2          CALL ADJUST2SOF (A,B);
```

AD-A102 693   NAVAL POSTGRADUATE SCHOOL MONTEREY CA   F/G 9/2
MICROCOMPUTER BASED SHIP-BOARD GUN CONTROL SYSTEM.(U)
MAR 81   A ERDOGAN

UNCLASSIFIED   NL

4 OF 7
AD
A102693

```
101   2        OUTPUT(OUT$OP$CODE) = MUL$CODE;
102   2        CALL CHECK;
103   2        CALL VAL$RESULT (C);
104   2        RETURN;
105   2        END MUL;


             /*********************************************************************
         -    ***************
             *
             * DIV:
             *   PROCEDURE USED TO PERFORM FIXED POINT DIVISION USING THE
             *   F.P. BOARD.
             *
             * PARAMETERS:
             *     - A.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH
         -     THE DIVI-
             *       DEND IS LOCATED.
             *    - B.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH
         -     THE DIVI-
             *       SOR IS LOCATED.
             *    - C.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH
         -     THE RESULT
             *       (QUOTIENT) IS DESIRED TO BE PLACED.
             *    - R.- POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH
         -     THE REMAIN-
             *       DER IS DESIRED TO BE PLACED. NOTICE THAT C AND R COULD
         -    POINT TO ANY
             *       OF THE TWO OPERANDS IF SO DESIRED.
             *
```

```
                 ***************************************************************
                  ***************/
126    1         DIV: PROCEDURE (A,B,C,R) PUBLIC;
127    2            DCL (A,B,C,R) ADDRESS;
128    2            CALL ADJUST2$OP(A,B);
129    2            OUTPUT(OUT$OP$CODE) = DIV$CODE;
110    2            CALL CHECK;
111    2            CALL VAL$RESULTS2(C,R);
112    2            RETURN;
113    2            END DIV;

                 /***************************************************************
                  *********
                  *
                  * FDIV:
                  *   PROCEDURE USED TO PERFORM EXTENDED FIXED POINT DIVISION U
                  SING THE
                  *   F.P. BOARD.
                  *
                  * PARAMETERS:
                  *   - A.- POINTER TO A 4 BYTE VECTOR THAT WILL PROVIDE THE DI
                  VIDEND.
                  *   - B.- POINTER TO A 2 BYTE VECTOR THAT WILL PROVIDE THE DI
                  VISOR.
                  *   - C.- POINTER TO A 4 BYTE VECTOR IN WHICH THE QUOTIENT WI
                  LL BE RETURNED.
                  *   - R.- POINTER TO A 4 BYTE VECTOR IN WHICH THE REMAINDER W
                  ILL BE RETURNED.
                  *
                  ***********************************************************
                -  **********/
114   1           EDIV: PROCEDURE (A,B,C,R) PUBLIC;
115   2              DCL (A,B,C,R) ADDRESS,
                         OP2 BASED P (2) BYTE,
                         I BYTE;
116   2              CALL ADJUST1$OF (A);
117   2              DO I = 2 TO LAST(OP2);
118   3                 RES$TABLE(I + 4) = OP2(I);
119   3                 END;
120   2              OUTPUT(OUT$OP$CODE) = EDIV$CODE;
121   2              CALL CHECK;
122   2              CALL VAL$RESULT1 (C,R);
123   2              RETURN;
124   2              END EDIV;


                  /**********************************************************
                -  *************
                  *
                  * FMUL:
                  *   PROCEDURE USED TO PERFORM FLOATING POINT MULTIPLICATION U
                -  SING THE
                  *   F.P. BOARD.
                  *
                  * PARAMETERS:
                  *   -A,B,C.- POINTERS TO 3 FOUR BYTE VECTORS THAT POINT TO TH
                -  E TWO OPERANDS
                  *    AND THE RESULT RESPECTIVELY. NOTE THAT THE RESULT COULD
                -  BE THE SAME
```

```
                 *     VECTOR USED TO INDICATE ANY OPERAND.
                 *
                 ************************************************************
               -  ***************/
125   1        FMUL: PROCEDURE (A,B,C) PUBLIC;
126   2           DCL (A,B,C) ADDRESS;
127   2           CALL ADJUST$OP(A,B);
128   2           OUTPUT(OUT$OP$CODE) = FMUL$CODE;
129   2           CALL CHECK;
130   2           CALL VAL$RESULT(C);
131   2           RETURN;
132   2           END FMUL;

                 /***********************************************************
               -  ****************
                 *
                 * FDIV:
                 *    PROCEDURE USED TO PERFORM FLOATING POINT DIVISION USING T
               -  HE F.P BOARD.
                 *
                 * PARAMETERS:
                 *    - A,B,C.- POINTERS TO 3 FOUR BYTE VECTORS THAT WILL POINT
               -   TO THE DIVIDEND
                 *       AND THE DIVISOR ON THE FIRST TWO, AND THE RESULT ON THE
               -   THIRD. NOTE
                 *       THAT THE RESULT COULD BE PUT IN THE SAME PLACE OF ANY O
               -  PERAND IF SO
                 *       DESIRED.
                 *
```

```
                    ********************************************************************
                -    ***************/
133   1         FDIV: PROCEDURE (A,B,C) PUBLIC;
134   2            DCL (A,B,C) ADDRESS;
135   2            CALL ADJUST$OP (A,B);
136   2            OUTPUT(OUT$OP$CODE) = FDIV$CODE;
137   2            CALL CHECK;
138   2            CALL VAL$RESULT(C);
139   2            RETURN;
140   2            END FDIV;


                /*******************************************************************
                -    **************
                *
                * FADD:
                *    PROCEDURE USED TO PERFORM FLOATING POINT ADDITION USING T
                -    HE F.P. BOARD.
                *
                * PARAMETERS:
                *    - A,B,C.- POINTERS TO 3 FOUR BYTE VECTORS. THE FIRST TWO
                -    POINT TO THE
                *        TWO OPERANDS, AND THE THIRD ONE POINTS TO THE VECTOR RE
                -    SULT. NOTE
                *        THAT THE RESULT COULD ALSO BE PLACED IN ANY OF THE OPER
                -    AND VECTORS.
                *
                ********************************************************************
                -    *************/
141   1         FADD: PROCEDURE (A,B,C) PUBLIC;
```

```
142   2         DCL (A,B,C) ADDRESS;
143   2         CALL ADJUST$OP (A,B);
144   2         OUTPUT(OUT$OF$CODE) = FADD$CODE;
145   2         CALL CHECK;
146   2         CALL VAL$RESULT (C);
147   2         RETURN;
148   2         END FADD;

               /*************************************************************
            -   **********
               *
               * FSUB:
               *   PROCEDURE USED TO PERFORM FLOATING POINT SUBTRACTION USIN
            -   G THE
               *   F.P. BOARD.
               *
               * PARAMETERS:
               *   - A,B,C.- POINTERS TO 3 FOUR BYTE VECTORS. THE FIRST TWO
            -   POINT TO
               *     BOTH OPERANDS, AND THE THIRD ONE POINTS TO THE VECTOR W
            -   HERE THE
               *     RESULT IS DESIRED TO BE PLACED. NOTE THAT THE RESULT CO
            -   ULD BE
               *     PLACED IN ANY OF BOTH OPERANDS.
               *
               **************************************************************
            -   *********/
149   1        FSUB: PROCEDURE (A,B,C) PUBLIC;
150   2          DCL (A,B,C) ADDRESS;
```

```
151   2          CALL ADJUST$OP (A,B);
152   2          OUTPUT(OUT$OP$CODE) = FSUB$CODE;
153   2          CALL CHECK;
154   2          CALL VAL$RESULT (C);
155   2          RETURN;
156   2          END FSUB;

               /*************************************************************
        -       *************
               *
               * FSQR:
               *   PROCEDURE USED TO PERFORM A FLOATING POINT SQUARE USING T
        -       HE F.P.
               *   BOARD.
               *
               * PARAMETERS:
               *   - A,C.- POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO A VE
        -       CTOR IN WHICH
               *     A FLOATING POINT QUANTITY IS LOCATED AND TO WHICH ITS S
        -       QUARE WILL BE
               *     OBTAINED. C POINTS TO A VECTOR IN WHICH THE RESULT IS D
        -       ESIRED TO BE
               *     PLACED. NOTE THAT A AND C COULD POINT TO THE SAME VECTO
        -       R.
               *
               **************************************************************
        -       ************/
157   1        FSQR: PROCEDURE (A,C)  PUBLIC;
158   2          DCL (A,C) ADDRESS;
```

```
 159   2          CALL ADJUST1$OP(A);
 160   2          OUTPUT(OUT$OP$CODE) = FSQR$CODE;
 161   2          CALL CHECK;
 162   2          CALL VAL$RESULT (C);
 163   2          RETURN;
 164   2          END FSQR;

                /*****************************************************************
           -     ***************
                *
                * FLTDS:
                *   PROCEDURE USED TO PERFORM A FIXED-TO-FLOAT CONVERSION USI
           -     NG THE F.P.
                *   BOARD.
                *
                * PARAMETERS:
                *    - A,C.- POINTERS TO 2 FOUR BITE VECTORS. A POINTS TO A VE
           -     CTOR CONTAINING
                *      A FIXED POINT INTEGER AND C POINTS TO A VECTOR WHERE TH
           -     E SINGLE PRE-
                *      CISION FLOATING POINT REPRESENTATION OF THE SAME VALUE,
           -      IS DESIRED TO
                *      BE PLACED. NOTE THAT BOTH, A AND C, COULD POINT TO THE
           -     SAME VECTOR.
                *
                *****************************************************************
           -      ***************/
 165   1        FLTDS: PROCEDURE (A,C) PUBLIC;
 166   2           DCL (A,C) ADDRESS;
 167   2          CALL ADJUST1$OP (A);
 168   2          OUTPUT(OUT$OP$CODE) = FLTDS$CODE;
 169   2          CALL CHECK;
 170   2          CALL VAL$RESULT (C);
 171   2          RETURN;
 172   2          END FLTDS;

               /**************************************************************
           -    ****************
               *
               * FIXSD:
               *   PROCEDURE USED TO PERFORM A FLOAT-TO-FIXED CONVERSION USI
           -    NG THE F.P.
               *   BOARD.
               *
               * PARAMETERS:
               *    - A,C.- POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO A VE
           -    CTOR CONTAINING
               *     THE FIXED POINT QUANTITY DESIRED TO BE CONVERTED. C POI
           -    NTS TO A VECTOR
               *     IN WHICE THE CONVERSION IS DESIRED TO BE PIACED. NOTE T
           -    HAT BOTH COULL
               *     POINT TO THE SAME VECTOR.
               *
               **************************************************************
           -    ****************/
 173   1       FIXSD: PROCEDURE (A,C) PUBLIC;
 174   2          DCL (A,C) ADDRESS;
 175   2          CALL ADJUST1$OP (A);
```

```
176   2         OUTPUT(OUT$OP$CODE) = FIXSD$CODE;
177   2         CALL CHECK;
178   2         CALL VAL$RESULT (C);
179   2         RETURN;
180   2         END FIXSD;

             /*****************************************************************
           -  ******************
              *
              * FSORT:
              *   PROCEDURE TO PERFORM THE SQUARE ROOT OF A FLOATING POINT
           -  NUMBER USING
              *   THE F.P. BOARD.
              *
              * PARAMETERS:
              *   - A,C.- POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO THE
           -  VECTOR CONTAINING
              *     THE NUMBER TO WHICH ITS SQUARE ROOT WILL BE OBTAINED. C
           -   POINTS TO THE
              *     VECTOR IN WHICH THE RESULT WILL BE PLACED. NOTE THAT BO
           -  TH POINTERS
              *     COULD BE REFERING TO THE SAME VECTOR.
              *
              *****************************************************************
           -  *****************/
181   1      FSQRT: PROCEDURE (A,C)  PUBLIC;
182   2         DCL (A,C) ADDRESS;
183   2         CALL ADJUST1$OF (A);
184   2         OUTPUT(OUT$OP$CODE) = FSQRT$CODE;
```

```
185    2          CALL CHECK;
186    2          CALL VAL$RESULT (C);
187    2          RETURN;
188    2          END FSQRT;


               /*****************************************************************
          -     ******************
               *
               * FCMPR:
               *    PROCEDURE USED TO COMPARE TWO FLOATING POINT NUMBERS USIN
          -     G THE F.P. BOARD.
               *
               * PARAMETERS:
               *    - A,B.- POINTERS TO 2 FOUR BYTE VECTORS CONTAINING THE TW
          -     O FLOATING
               *       POINT NUMBERS DESIRED TO BE COMPARED.
               *    - C.- POINTS TO A BYTE VALUE CONTAINING THE CODE CORRESPO
          -     NDING TO THE
               *       TYPE OF COMPARISON DESIRED:
               *                    <   ..... 0
               *                    <=  ..... 1
               *                    >   ..... 2
               *                    >=  ..... 3
               *                    =   ..... 4
               *                    <>  ..... 5
               *
               * USAGE:
               *    TYPED PROCEDURE. IF THE RELATION DESIRED TO BE TESTED HOL
```

```
              -  IS, A VALUE OF
              *   '01H' (TRUE) IS RETURNED, OTHERWISE A VALUE OF '00H' (FAL
              -  SE) WILL BE
              *   RETURNED.
              *
              ************************************************************
              -  ****************/
 189    1     FCMPR: PROCEDURE (A,B,C) BYTE PUBLIC;
 190    2        DCL (A,B,C) ADDRESS,
                     (TEST BASED C,RESULT, FLAG) BYTE;
 191    2        FLAG = 000H;         /* RESET FLAG USED TO CHECK TWO NEGATIVE
              -     NUMBERS. */
 192    2        CALL ADJUST$OP (A,B);
 193    2        IF (RES$TABLE(3) >= 80H) AND
                     (RES$TABLE(7) >= 80H)
                     THEN FLAG = 0FFH;
 195    2        OUTPUT(OUT$OP$CODE) = FCMPR$CODE;
 196    2        CALL CHECK;
 197    2        RESULT = INPUT (IN$STATUS) AND 0E0H;
 198    2        IF FLAG AND (RESULT <> 80H)
                 THEN DO;
 200    3              IF RESULT = 40H
                          THEN RESULT = 20H;
                          ELSE RESULT = 40H;
 202    3              END;
 203    3        RETURN (RESULT:= COMPARE(RESULT,TEST));
 204    2        END FCMPR;
 205    2
```

```
     /*****************************************************************
   -  ****************
     *
     * FZTST:
     *   PROCEDURE USED TO TEST A FLOATING POINT NUMBER AGAINST 0.
   -  0 USING THE F.P.
     *   BOARD.
     *
     * PARAMETERS:
     *   - A.- POINTER TO A FOUR BYTE VECTOR CONTAINING THE FLOATI
   -  NG POINT VALUE
     *     DESIRED TO BE TESTED.
     *   - C.- POINTER TO A BYTE VALUE CONTAINING THE CODE OF THE
   -  COMPARISON DESI-
     *     RED, ACCORDING TO THE FOLLOWING RULES:
     *                      <  ..... 0
     *                      <= ..... 1
     *                      >  ..... 2
     *                      >= ..... 3
     *                      =  ..... 4
     *                      <> ..... 5
     *
     * USAGE:
     *   TYPED PROCEDURE. IF THE RELATION DESIRED TO BE TESTED HOL
   -  DS, A VALUE OF
     *   '21H' (TRUE) IS RETURNED, OTHERWISE A VALUE OF '00H' (FAL
   -  SE) WILL BE RE-
     *   TURNED.
     *
     *****************************************************************
```

```
                -   ****************/
206   1          FZTST: PROCEDURE (A,C) BYTE PUBLIC;
207   2            DCL (A,C) ADDRESS,
                       OP1 BASED A (4) BYTE,
                       (TEST BASED C, RESULT, I) BYTE;
208   2            DCL LOWER$BOUND (4) BYTE DATA (0E2H,0FAH,0FFF,0E4H), /* -0
                -   .0000229 */
                       UPPER$BOUND (4) BYTE DATA (0E2H,0FAH,0FFH,034H), /* 0.
                -   0FF2229 */
                       LOW BYTE DATA (01H),    /* LESS THAN OR ECUAL */
                       HIGH BYTE DATA (03H);   /* GREATER THAN OR EQUAL */

                       /* CHECK IF NUMBER IS IN BOUNDARIES OF DEFINED SYSTEM Z
                -   ERO */
209   2            IF (TEST <> 0) AND (TEST <> 2) AND
                       (FCMPR(.OP1, .LOWER$BOUND, .HIGH)) AND
                       (FCMPR(.OP1, .UPPER$BOUND, .LOW))
                   THEN DO;
211   3                DO I = 0 TO 3;
212   4                   OP1(I) = 00H;
213   4                   END;
214   3                END;
215   2            CALL ADJUST1$OF (A);
216   2            OUTPUT(OUT$OP$CODE) = FZTST$CODE;
217   2            CALL CHECK;
218   2            RESULT = INPUT (INSSTATUS) AND 0E0H;
219   2            RETURN (RESULT:= COMPARE(RESULT,TEST));
220   2            END FZTST;

                 /*****************************************************************
```

```
        -   ****************
            *
            * EXCH:
            *   PROCEDURE USED TO EXCHANGE TWO FLOATING POINT VALUES USIN
        -   G THE F.P.
            *   BOARD.
            *
            * PARAMETERS:
            *   - A,C.- POINTERS TO 2 FOUR BYTE VECTORS CONTAINING TWO FL
        -   OATING
            *     POINT NUMBERS DESIRED TO BE EXCHANGED.
            *
            *****************************************************************
        -   ****************/
221   1     EXCH: PROCEDURE (A,C) PUBLIC;
222   2        DCL (A,C) ADDRESS;
223   2        CALL ADJUST$OP (A,C);
224   2        OUTPUT(OUT$OP$CODE) = EXCH$CODE;
225   2        CALL CHECK;
226   2        CALL VAL$RESULT$1(A,C);
227   2        RETURN;
228   2        END EXCH;


            /****************************************************************
        -   ***************
            *
            * COS$SIN:
            *   THIS PROCEDURE IS USED TO CALCULATE THE COSINE AND SINE F
```

```
                -   UNCTIONS
                *     OF A GIVEN ARGUMENT IN RADIANS.
                *
                * PARAMETERS:
                *     - A.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE FLOATING
                -    POINT REPRE-
                *        SENTATION OF AN ANGLE IN RADIANS IS LOCATED.
                *
                *     - C.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF
                -    THE COSINE
                *        OF THE GIVEN ANGLE, WILL BE PLACED.
                *
                *     - S.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF
                -    THE SINE OF
                *        THE GIVEN ANGLE WILL BE PLACED.
                *
                ****************************************************************
                -    **************/
 229   1        COS$SIN: PROCEDURE (A,C,S) PUBLIC;
 230   2           DCL (A,C,S) ADDRESS,
                         ANGLE BASED A (4) BYTE,
                         COSINE BASED C (4) BYTE,
                         SINE BASED S (4) BYTE,
                         ANGLE$SC (4) BYTE,
                         TEMP (4) BYTE,
                         TEMP0 (4) BYTE,
                         TEMP1 (4) BYTE,
                         MINUS$ONE (4) BYTE DATA (00H,00H,80H,0BFH),   /* -1.0
                -    */
                         ONE$FLOAT (4) BYTE DATA (00H,00H,80H,3FH),    /* 1.0 I
```

```
-    A F.P. FORMAT.*/
          TWO$FLOAT (4) BYTE DATA (00H,00H,00H,40H),      /* 2.0 *
-    /
          PI$FLOAT (4) BYTE DATA (0DBH,0FH,49H,40H),      /* 3.141
-    593 */
          TWO$PI (4) BYTE DATA (0DBH,0FH,0C9H,40H),       /* 6.283
-    1853 */
          PI$OVER2 (4) BYTE DATA (0DBH,0FH,0C9H,3FH),     /* 1.570
-    7963 */
          PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H),   /* 4.712
-    3889 */
          CONST$1 (4) BYTE DATA (0E5H,1FH,12H,3FH),       /* 0.570
-    7963 */
          CONST$2 (4) BYTE DATA (0B6H,5DH,25H,0BFH),      /* -0.64
-    5964 */
          CONST$3 (4) BYTE DATA (0E1H,35H,0A3H,3DH),      /* 0.079
-    629261 */
          CONST$4 (4) BYTE DATA (0AAH,68H,99H,0BBH),      /* -0.00
-    4681656B */
          CONST$5 (4) BYTE DATA (25H,0BH,28H,39H),        /* 0.000
-    1602558 */
          CONST$6 (4) BYTE DATA (0A4H,067H,66H,0B6H),     /* -0.00
-    00034333 */
          CHECK BYTE DATA (002H),     /* CHECK FOR GREATER THAN *
-    /
          CHECK1 BYTE DATA (003H),    /* CHECK FOR GREATER THAN O
-    R EQUAL */
          CHECK2 BYTE DATA (001H),    /* CHECK FOR LESS THAN OR E
-    QUAL */
          CHECK3 BYTE DATA (004H),    /* CHECK FOR EQUAL */
```

```
                     (SIGN,SIGN1,QUAD,I) BYTE;

  2            DO I = Ø TO 3;
  3               TEMPØ(I) = ANGLE(I);
  3               SINE(I), COSINE(I) = ØØH;
  3               END;
             /*   CHECK IF ANGLE IS >= 36Ø DEGREES.    */
  2            SIGN = FCMPR(.TEMPØ,.TWO$PI,.CHECK1);
  2            DO WHILE SIGN;
  3               CALL FSUB(.TEMPØ,.TWO$PI,.TEMPØ);
  3               SIGN = FCMPR(.TEMPØ,.TWO$PI,.CHECK1);
  3               END;
             /*   CHECK IF ANGLE IS NEGATIVE.    */
  2            DO WHILE TEMPØ(3) >= Ø8ØH;
  3               CALL FADD(.TEMPØ,.TWO$PI,.TEMPØ);
  3               END;
             /*   CHECK FOR SPECIAL CASES    */
  2            IF FCMPR (.TEMPØ, .PI$OVER2, .CHECK3)
               THEN DO;     /* 9Ø DEGREES */
  3               DO I = Ø TO 3;
  4                  SINE(I) = ONE$FLOAT(I);
  4                  END;
  3               RETURN;
  3               END;
  2            IF FCMPR (.TEMPØ, .PI$3$OVER2, .CHECK3)
               THEN DO;     /* 27Ø DEGREES */
  3               DO I = Ø TO 3;
  4                  SINE(I) = MINUS$ONE(I);
  4                  END;
  3               RETURN;
```

```
      END;
   IF FCMPR (.TEMP0, .TWO$PI, .CHECK3)
   THEN DO;     /* 360 DEGREES */
      DO I = 0 TO 3;
         COSINE(I) = ONE$FLOAT(I);
         END;
      RETURN;
      END;
/*   TO NORMALIZE THE ANGLE BETWEEN 0 AND 90 DEGREES.    */
   QUAD = 1;
   IF (SIGN := FCMPR(.TEMP0,.PI$OVER2,.CHECK)) AND
      (SIGN1 := FCMPR(.TEMP0,.PI$FLOAT,.CHECK2))
   THEN DO;
      QUAD = 2;
      CALL FSUB(.PI$FLOAT,.TEMP0,.TEMP0);
      END;
   ELSE DO;
      IF (SIGN := FCMPR(.TEMP0,.PI$FLOAT,.CHECK)) AND
         (SIGN1 := FCMPR(.TEMP0,.PI$3$OVER2,.CHECK2))
      THEN DO;
         QUAD = 3;
         CALL FSUB(.TEMP0,.PI$FLOAT,.TEMP0);
         END;
      ELSE DO;
         IF (SIGN := FCMPR(.TEMP0,.PI$3$OVER2,.CHECK))
         THEN DO;
            QUAD = 4;
            CALL FSUB(.TWO$PI,.TEMP0,.TEMP0);
            END;
         END;
```

```
 283   3              END;
               /*   CONVERT ANGLE IN RADIANS TO ANGLE IN SEMICIRCLE UNITS.
          -     */
 284   2           CALL FDIV(.TEMP0,.PI$FLOAT,.TEMP);
               /*   GET THE SQUARE OF THE ANGLE.   */
 285   2           CALL FSQR(.TEMP,.ANGLE$SQ);
               /*   PERFORM HASTINGS APPROXIMATION.   */
 286   2           CALL FMUL(.ANGLE$SQ,.CONST$6,.TEMP1);
 287   2           CALL FADD(.TEMP1,.CONST$5,.TEMP1);
 288   2           CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
 289   2           CALL FADD(.TEMP1,.CONST$4,.TEMP1);
 290   2           CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
 291   2           CALL FADD(.TEMP1,.CONST$3,.TEMP1);
 292   2           CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
 293   2           CALL FADD(.TEMP1,.CONST$2,.TEMP1);
 294   2           CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
 295   2           CALL FADD(.TEMP1,.CONST$1,.TEMP1);
 296   2           CALL FMUL(.TEMP,.TEMP1,.TEMP1);
 297   2           CALL FADD(.TEMP1,.TEMP,.TEMP1);
 298   2           CALL FSQR(.TEMP1,.TEMP1);
 299   2           CALL FMUL(.TEMP1,.TWO$FLOAT,.TEMP1);
               /*   TO COMPUTE THE VALUE OF THE COSINE.   */
 300   2           CALL FSUB(.ONE$FLOAT,.TEMP1,.COSINE);
               /*   TO COMPUTE THE VALUE OF THE SINE.   */
 301   2           CALL FSQR(.COSINE,.TEMP1);
 302   2           CALL FSUB(.ONE$FLOAT,.TEMP1,.TEMP1);
 303   2           CALL FSQRT(.TEMP1,.SINE);
               /*   GIVE SIGNS TO COSINE AND SINE VALUES ACCORDING TO QUADRA
          -     NT   */
 304   2           DO CASE QUAD;
```

```
325   3               ;                               /*   FIRST QUADRANT
326   3               ;
              -     */
                      COSINE(3) = COSINE(3) XOR 80H;  /*   SECOND CUADRAN
327   3
              -   T   */
328   3               DO;
329   4                  COSINE(3) = COSINE(3) XOR 80H;  /*   THIRD CUADRANT
              -     */
310   4                  SINE(3) = SINE(3) XOR 80H;
311   4               END;
312   3               SINE(3) = SINE(3) XOR 80H;      /*   FOURTH QUADRAN
              -   T   */
313   3             END;
                 /*   ALL DONE. RETURN.   */
314   2            END COS$SIN;


                 /*************************************************************
              -   *******************
                  *
                  * ARC$TAN:
                  *   THIS PROCEDURE IS USED TO CALCULATE THE ARC$TAN FUNCTION
              -   IN RADIANS
                  *   GIVEN AS ARGUMENT A RATIO OF TWO VALUES IN FP REPRESENTAT
              -   ION.
                  *
                  * PARAMETERS:
                  *   - X.- POINTER TO A FOUR BYTE VECTOR REPRESENTING THE DENO
              -   MINATOR
```

```
                 *      OF THE RATIO.
                 *    - Y.- POINTER TO A FOUR BYTE VECTOR REPRESENTING THE NUME
               -   RATOR
                 *      OF THE RATIO.
                 *    - A.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF
               -    THE ANGLE
                 *      (IN RADIANS) WILL BE PLACED AFTER CALCULATION OF THE AR
               -   CSTAN.
                 *
                 ****************************************************************
               -  *****************/
315   1          ARC$TAN: PROCEDURE (X,Y,A) PUBLIC;
316   2             DCL (X,Y,A) ADDRESS,
                        DELTA$X BASED X (4) BYTE,
                        DELTA$Y BASED Y (4) BYTE,
                        ANGLE BASED A (4) BYTE,
                        PI$FLOAT (4) BYTE DATA (ØDBH,ØFH,49H,4ØH),       /* 3.
               -   141593 */
                        TWO$PI (4) BYTE DATA (ØDBH,ØFH,ØC9H,4ØH),        /* 6.
               -   2831853 */
                        PI$OVER4 (4) BYTE DATA (ØDBH,ØFH,49H,3FH),       /* Ø.
               -   78539819 */
                        PI$OVER2 (4) BYTE DATA (ØDBH,ØFH,ØC9H,3FH),      /* 1.
               -   5707963 */
                        PI$3$OVER2 (4) BYTE DATA (ØE4H,ØCBH,96H,4ØH),    /* 4.
               -   7123889 */
                        CONST$1 (4) BYTE DATA (ØF5H,ØFFH,7FH,3FH),       /* Ø.
               -   9999993329 */
                        CONST$2 (4) BYTE DATA (1CH,ØA5H,ØAAH,ØBEH),      /* -Ø
               -   .3332985605 */
```

```
                    [illegible]   BYTE DATA (ØA7H,4ØH,4CH,3EH),        /* Ø.
         -   1.4465[illegible] */
                    CONST$4 (4) BYTE DATA (63H,6CH,ØEH,ØBEH),          /* -.
         -   .1390853351 */
                    CONST$5 (4) BYTE DATA (ØDEH,77H,ØC5H,3DH),         /* Ø.
         -   Ø9642ØØ441 */
                    CONST$6 (4) BYTE DATA (ØC4H,Ø1H,65H,ØBDH),         /* -Ø
         -   .Ø559Ø98861 */
                    CONST$7 (4) BYTE DATA (51H,16H,ØE3H,3CH),          /* Ø.
         -   Ø2146122[illegible] */
                    CONST$8 (4) BYTE DATA (ØE6H,ØD7H,84H,ØBBH),        /* -Ø
         -   .ØØ4254Ø580 */
                    CHECK BYTE DATA (Ø4H),
                    MSG1(*) BYTE DATA
                         (' ARC$TAN FUNCTION UNDEFINED FOR BOTH ARGUMENTS
         -   EQUAL TO ZERO.$$'),
                    MSG2(*) BYTE DATA (' FATAL ERROR.$$'),
                    Z (4) BYTE,
                    Z$SQUARE (4) BYTE,
                    TEMP (4) BYTE,
                    TEMP1 (4) BYTE,
                    (SIGN$X, SIGN$Y, ZERO$X, ZERO$Y, I) BYTE;
 317   2        SIGN$X, SIGN$Y = ØØH;
 318   2        DO I = Ø TO 3;
 319   3           TEMP(I) = DELTA$Y(I);
 32Ø   3           TEMP1(I) = DELTA$X(I);
 321   3           END;
              /*   SAVE SIGN TO DETERMINE QUADRANT    */
 322   2        IF TEMP(3) >= 8ØH THEN SIGN$Y = ØFFH;
 324   2        IF TEMP1(3) >= 8ØH THEN SIGN$X = ØFFH;
```

```
                    /*   CHECK FOR VALID ARGUMENTS    */
 326   2               IF (ZEROSY := F2TST(.DELTASY,.CHECK)) AND
                          (ZEROSX := F2TST(.DELTASX,.CHECK))
                       THEN DO;
 328   3                   CALL CRT$PRINT$STRING (.MSG1);
 329   3                   CALL CRT$PRINT$STRING (.MSG2);
 330   3                   HALT;
 331   3                   END;
 332   2               IF ZEROSX
                       THEN DO;
 334   3                   IF SIGN$Y
                           THEN DO;
 336   4                       DO I = 0 TO 3;
 337   5                          ANGLE(I) = PI$3$OVER2(I);
 338   5                          END;
 339   4                       RETURN;
 340   4                       END;
 341   3                   ELSE DO;
 342   4                       DO I = 0 TO 3;
 343   5                          ANGLE(I) = PI$OVER2(I);
 344   5                          END;
 345   4                       RETURN;
 346   4                       END;
 347   3                   END;
                    /*   FORM Z TO PERFORM HASTINGS APPROXIMATION    */
 348   2               TEMP(3) = TEMP(3) AND 7FH;
 349   2               TEMP1(3) = TEMP1(3) AND 7FH;
 350   2               CALL FSUB(.TEMP, .TEMP1, .Z);
 351   2               CALL FADD(.TEMP, .TEMP1, .TEMP);
 352   2               CALL FDIV(.Z, .TEMP, .Z);
 353   2         CALL FSQR(.Z, .Z$SQUARE);
               /*   PERFORM HASTINGS APPROXIMATION FOR ARC$TAN    */
 354   2         CALL FMUL(.Z$SQUARE, .CONST$8, .TEMP);
 355   2         CALL FADD(.TEMP, .CONST$7, .TEMP);
 356   2         CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
 357   2         CALL FADD(.TEMP, .CONST$6, .TEMP);
 358   2         CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
 359   2         CALL FADD(.TEMP, .CONST$5, .TEMP);
 360   2         CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
 361   2         CALL FADD(.TEMP, .CONST$4, .TEMP);
 362   2         CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
 363   2         CALL FADD(.TEMP, .CONST$3, .TEMP);
 364   2         CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
 365   2         CALL FADD(.TEMP, .CONST$2, .TEMP);
 366   2         CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
 367   2         CALL FADD(.TEMP, .CONST$1, .TEMP);
 368   2         CALL FMUL(.Z, .TEMP, .TEMP);
 369   2         CALL FADD(.TEMP, .PI$OVER4, .ANGLE);
               /*   RESTORE ANGLE TO PROPER QUADRANT   */
 370   2         IF (NOT SIGN$Y) AND SIGN$X             /* SECOND QUADRANT *
             -  /
                   THEN CALL FSUB(.PI$FLOAT, .ANGLE, .ANGLE);
 372   2         IF SIGN$Y AND SIGN$X                   /* THIRD QUADRANT */
                   THEN CALL FADD(.PI$FLOAT, .ANGLE, .ANGLE);
 374   2         IF SIGN$Y AND (NOT SIGN$X)             /* FOURTH QUADRANT *
             -  /
                   THEN CALL FSUB(.TWO$PI, .ANGLE, .ANGLE);
               /*   ALL DONE. RETURN.   */
 376   2         END ARC$TAN;
 377   1      END FLOATING$POINT;


MODULE INFORMATION:

     CODE AREA SIZE     = 0D1CH   3356D
     VARIABLE AREA SIZE = 00A4H    164D
     MAXIMUM STACK SIZE = 000EH     14D
     939 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER                                   20 MAR 81   PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE COMMANDS
OBJECT MODULE PLACED IN CMDS1.OBJ
COMPILER INVOKED BY:  :F1:PLM80 CMDS1.SRC PAGELENGTH(33) PAGEWIDTH(75) DATE
                      -(20 MAR 81)



  1           COMMANDS: DO;


  2   1       CRT$READ:
                 PROCEDURE BYTE EXTERNAL;
  3   2          END;

  4   1       CRT$PRINT$STRING:
                 PROCEDURE (A) EXTERNAL;
  5   2          DECLARE A ADDRESS; END;

  7   1       CRT$WRITE:
                 PROCEDURE (A) EXTERNAL;
  8   2          DECLARE A BYTE; END;

 10   1       SEND$CR:
                 PROCEDURE EXTERNAL;
 11   2          END;

 12   1       SEND$BEL:
                 PROCEDURE EXTERNAL;
 13   2          END;

 14   1        SEND$BS:
                 PROCEDURE EXTERNAL;
 15   2          END;

 16   1        SEND$SUB:
                 PROCEDURE EXTERNAL;
 17   2          END;

 18   1        SEND$CRLF:
                 PROCEDURE EXTERNAL;
 19   2          END;

 20   1        GET$STRING:
                 PROCEDURE (A,B) EXTERNAL;
 21   2          DECLARE A ADDRESS,B BYTE; END;

 23   1        PUT$NUMBER$BUFFER:
                 PROCEDURE (A,B) EXTERNAL;
 24   2          DECLARE A BYTE, B ADDRESS; END;

 26   1        ASCII$TO$FLOAT:
                 PROCEDURE (A,B,C) EXTERNAL;
 27   2          DECLARE (A,C) ADDRESS, B BYTE; END;

 29   1        FLOAT$TO$ASCII:
                 PROCEDURE (A,B,C) EXTERNAL;
 30   2          DECLARE (A,B,C) ADDRESS; END;
```

```
  32    1       CLEAR$LOW$SCREEN:
                   PROCEDURE EXTERNAL;
  33    2          END;

  34    1       FADD:
                   PROCEDURE (A,B,C) EXTERNAL;
  35    2          DECLARE (A,B,C) ADDRESS; END;

  37    1       FSUB:
                   PROCEDURE (A,B,C) EXTERNAL;
  38    2          DECLARE (A,B,C) ADDRESS; END;

  40    1       FMUL:
                   PROCEDURE (A,B,C) EXTERNAL;
  41    2          DECLARE (A,B,C) ADDRESS; END;

  43    1       FDIV:
                   PROCEDURE (A,B,C) EXTERNAL;
  44    2          DECLARE (A,B,C) ADDRESS; END;

  46    1       EDIV:
                   PROCEDURE (A,B,C,D) EXTERNAL;
  47    2          DECLARE (A,B,C,D) ADDRESS; END;

  49    1       FLTDS:
                   PROCEDURE (A,B) EXTERNAL;
  50    2          DECLARE (A,B) ADDRESS; END;

  52    1       FIXSD:
                   PROCEDURE (A,B) EXTERNAL;
```

```
 53   2          DECLARE (A,B) ADDRESS; END;

 55   1        FCMPR:
                 PROCEDURE (A,B,C) BYTE EXTERNAL;
 56   2          DECLARE (A,B) ADDRESS, C BYTE; END;

 58   1        FZTST:
                 PROCEDURE (A,B) BYTE EXTERNAL;
 59   2          DECLARE (A,B) ADDRESS; END;


 61   1        DECLARE LIT LITERALLY 'LITERALLY',
                       DCL LIT 'DECLARE';

 62   1        DCL POINT LIT '2EH',                        /* DECIMAL POINT. */
                   ETEOL LIT '17H';                        /* ERASE TO END OF LI
             -  NE. */

 63   1        DCL
                 FP$0$25 (4) BYTE DATA (00H,00H,80H,3EH),
                 FP$2 (4) BYTE DATA (00H,00H,00H,40H),
                 FP$5 (4) BYTE DATA (00H,00H,0A0H,040H),
                 FP$25 (4) BYTE DATA (00H,00H,0C8H,41H),
                 FP$60 (4) BYTE DATA (00H,00H,70H,42H),
                 FP$90 (4) BYTE DATA (00H,00H,0B4H,42H),
                 FP$100 (4) BYTE DATA (00H,00H,0C8H,042H),
                 FP$180 (4) BYTE DATA (00H,00H,34H,43H),
                 FP$360 (4) BYTE DATA (0DFH,0FFH,0B3H,043H),
                    FP$2000 (4) BYTE DATA (0E4H,2BH,0FDH,044H),            /* 20
             -   25.3716 */
                    FP$202500 (4) BYTE DATA (04AH,0CAH,045H,048H),         /* 20
             -   2537.15625 */
                    FP$59$9 (4) BYTE DATA (9AH,99H,6FH,42H),
                    FP$99$9 (4) BYTE DATA (0CDH,0CCH,0C7H,042H),
                    FP$MIN$TO$RAD (4) BYTE DATA (98H,82H,98H,39H);        /* 0.0
             -   0029889 */


                 /*****************************************************************
             -    *************
                 *
                 * PRINT$ERROR$MSG:
                 *   THIS PROCEDURE IS USED TO PRINT AN ERROR MESSAGE FOR INVA
             -    LID INPUT.
                 *   IT WILL ALSO RETURN THE CURSOR TO THE BEGINNING OF THE SA
             -    ME LINE.
                 *
                 ******************************************************************
             -    *************/
  64   1         PRINT$ERROR$MSG: PROCEDURE;
  65   2            DCL MSG(*) BYTE DATA (' ***  BAD FORMAT.  ***$$');
  66   2            CALL SEND$BEL;
  67   2            CALL CRT$PRINT$STRING(.MSG);
  68   2            CALL SEND$CR;
  69   2            CALL SEND$SUB;
  70   2            END PRINT$ERROR$MSG;
```

```
                  /************************************************************
               -   **************
                  *
                  * CHECK$YES$NO:
                  *   PROCEDURE USED TO CHECK FOR A VALID YES/NO INPUT FROM THE
               -    CRT.
                  *
                  * USAGE:
                  *   TYPED PROCEDURE. A VALUE OF 1 WILL BE RETURNED IF THE ANS
               -   WER IS 'YES',
                  *   OTHERWISE, THE VALUE RETURNED IS 0.
                  *
                  ************************************************************
               -   ***************/
  71   1          CHECK$YES$NO: PROCEDURE BYTE PUBLIC;
  72   2             DCL CHAR BYTE;
  73   2             CHAR = CRT$READ;
  74   2             DO WHILE (CHAR <> 'Y') AND (CHAR <> 'N')      /* UPPER CASE
               -   . */
                         AND (CHAR <> 'y') AND (CHAR <> 'n');     /* LOWER CASE
               -   . */
  75   3                CALL SEND$BEL;
  76   3                CHAR = CRT$READ;
  77   3                END;
  78   2             IF (CHAR = 'Y') OR (CHAR = 'y')
                     THEN DO;
  80   3                 CALL CRT$PRINT$STRING(.('YES$$'));
  81   3                 RETURN 1;
```

```
 82    3              END;
 83    2          ELSE DO;
 84    3              CALL CRT$PRINT$STRING(.('NO $$'));
 85    3              RETURN 0;
 86    3              END;
 87    2           END CHECK$YES$NO;


                /*************************************************************
            -    **************
                 *
                 * CHECK$FP$VALUE:
                 *   THIS PROCEDURE IS USED TO CHECK A GIVEN VALUE AGAINST A G
            -    IVEN LIMIT.
                 *
                 * PARAMETERS:
                 *    - A.- POINTER TO A FP VALUE.
                 *    - B.- POINTER TO A FP VALUE DENOTING THE LIMIT.
                 *
                 * USAGE:
                 *   TYPED PROCEDURE. A VALUE OF 00H WILL BE RETURNED IF THE V
            -    ALUE IS GREATER
                 *   THAN THE GIVEN LIMIT. OTHERWISE A VALUE OF 001H WILL BE R
            -    ETURNED.
                 *
                 *************************************************************
            -    ****************/
 88    1        CHECK$FP$VALUE: PROCEDURE (A,B) BYTE PUBLIC;
 89    2          DCL (A,B) ADDRESS,
                        (CHECK, TWO) BYTE;
  90    2           TWO = 2;
  91    2           IF (CHECK:= FCMPR(A,B,.TWO))
                    THEN DO;
  93    3                CALL PRINT$ERROR$MSG;
  94    3                RETURN 0;
  95    3                END;
  96    2           ELSE DO;
  97    3                CALL CRT$WRITE(ETEOL);                      /* ERASE TO THE END
             -    OF LINE. */
  98    3                RETURN 1;
  99    3                END;
 100    2           END CHECK$FP$VALUE;


                /*************************************************************
             -   **************
                 *
                 * CHECK$INPUT:
                 *   PROCEDURE USED TO CHECK THE VALIDITY OF THE INPUT PRESENT
             -    AT THAT MOMENT
                 *   IN THE SCREEN.
                 *
                 *************************************************************
             -   ****************/
 101    1       CHECK$INPUT: PROCEDURE BYTE PUBLIC;
 102    2          DCL CHAR BYTE;
 103    2          CALL CRT$PRINT$STRING(.('IS THE INPUT CORRECT? (Y/N)  $$')
             -   );
```

```
104   2          CHAR = CHECK$YES$NO;
105   2          RETURN CHAR;
106   2          END CHECK$INPUT;


                /*************************************************************
         -       *****************
                *
                * GET$DEGREES:
                *   THIS PROCEDURE IS USED TO OBTAIN 2 OR 3 NUMERIC CHARACTER
         -       S FROM THE CRT
                *   THAT REPRESENT A VALUE IN DEGREES OF LATITUDE OR LONGITUD
         -       E RESPECTIVELY.
                *
                * PARAMETERS:
                *    - NUM.- NUMBER OF NUMERIC CHARACTERS DESIRED. (2 OR 3 ONL
         -       Y)
                *    - A.- POINTER TO A MEMORY LOCATION IN WHICH THE RESULT IS
         -        DESIRED.
                *
                * USAGE:
                *   THE FLOATING POINT REPRESENTATION OF THE DEGREES, WILL BE
         -        RETURNED CON-
                *   VERTED TO MINUTES OF LAT OR LONG.
                *
                **************************************************************
         -       **************/
107   1         GET$DEGREES: PROCEDURE (NUM,A) PUBLIC;
108   2            DCL A ADDRESS,
                      NUMBER BASED A (4) BYTE,
                      (NUM, OK) BYTE,
                      BUFFER(6) BYTE;
 109   2          BUFFER(0), BUFFER(1) = NUM;
 110   2          BUFFER(NUM + 2) = 0;
 111   2          OK = 0;
 112   2          DO WHILE OK = 0;
 113   3             CALL CRT$PRINT$STRING(.('DEGREES:  $$'));
 114   3             CALL PUT$NUMBER$BUFFER(NUM,.BUFFER(2));
 115   3             CALL ASCII$TO$FLOAT(.BUFFER,NUM + 3,.NUMBER);
 116   3             IF NUM = 2
                        THEN OK = CHECK$FP$VALUE(.NUMBER,.FP$90);
 118   3                ELSE OK = CHECK$FP$VALUE(.NUMBER,.FP$180);
 119   3             END;
 120   2          CALL FMUL(.NUMBER,.FP$60,.NUMBER);
 121   2          END GET$DEGREES;


            /*****************************************************************
        -    *****************
             *
             * GET$MINUTES:
             *    THIS PROCEDURE IS USED TO GET FROM THE CRT, THREE NUMERIC
        -     CHARACTERS RE-
             *    PRESENTING THE VALUE OF MINUTES. THIS PROCEDURE WILL PROM
        -    PT FOR TWO
             *    INTEGERS AND ONE DECIMAL VALUE.
             *
             * PARAMETERS:
                  *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRES
                -  ENTATION OF THE
                  *     CHARACTERS OBTAINED, IS DESIRED TO BE PLACED.
                  *
                  *****************************************************************
                -  *****************/
 122    1         GET$MINUTES: PROCEDURE (A) PUBLIC;
 123    2            DCL A ADDRESS,
                         NUMBER BASED A (4) BYTE,
                         BUFFER(6) BYTE,
                         OK BYTE;
 124    2            BUFFER(0) = 3;
 125    2            BUFFER(1) = 2;
 126    2            BUFFER(5) = 0;
 127    2            OK = 0;
 128    2            DO WHILE OK = 0;
 129    3               CALL CRT$PRINT$STRING(.('MINUTES:  $$'));
 130    3               CALL PUT$NUMBER$BUFFER(2,.BUFFER(2));
 131    3               CALL CRT$WRITE(POINT);
 132    3               CALL PUT$NUMBER$BUFFER(1,.BUFFER(4));
 133    3               CALL ASCII$TO$FLOAT(.BUFFER,6,.NUMBER);
 134    3               OK = CHECK$FP$VALUE(.NUMBER,.FP$59$9);
 135    3               END;
 136    2            END GET$MINUTES;


                  /****************************************************************
                -  ***************
                  *
                  * GET$SIGN:
                  *    THIS PROCEDURE IS USED TO GET FROM THE CRT, A CHARACTER T
                -  HAT WILL REPRE-
                  *    SENT THE N/S LATITUDE OR E/W LONGITUDE.
                  *
                  * PARAMETERS:
                  *    - T1,T2.- ASCII CHARACTERS REPRESENTING THE VALUES AGAINS
                -  T WHICH THE
                  *       INPUT FROM THE CRT IS DESIRED TO BE COMPARED.
                  *
                  * USAGE:
                  *    TYPED PROCEDURE. A VALUE OF 1 WILL BE RETURNED IF THE INP
                -  UT FROM THE CRT
                  *    IS EQUAL TO THE VALUE OF T1, OTHERWISE A VALUE OF 0 WILL
                -  BE RETURNED.
                  *
                  *************************************************************
                -  ************/
 137   1          GET$SIGN: PROCEDURE (T1,T2) BYTE PUBLIC;
 138   2              DCL (T1,T2,SIGN) BYTE;
 139   2              SIGN = 0;
 140   2              DO WHILE (SIGN <> T1) AND (SIGN <> T2);
 141   3                  CALL CRT$PRINT$STRING(.('SIGN:  $$'));
 142   3                  CALL GET$STRING(.SIGN,1);
 143   3                  IF (SIGN <> T1) AND (SIGN <> T2)
                          THEN CALL PRINT$ERROR$MSG;
 145   3                  ELSE CALL CRT$WRITE(ETBOL);
 146   3                  END;
 147   2              IF SIGN = T1 THEN RETURN 1;
 149   2                         ELSE RETURN 0;
 150   2          END GET$SIGN;

                 /*************************************************************
          -      **********
                 *
                 * FP$FORMAT:
                 *   THIS PROCEDURE IS USED TO OBTAIN AN SPECIFIED NUMBER OF A
          -      SCII CHARAC-
                 *   TERS REPRESENTING A NUMERIC VALUE IN FP FORMAT.
                 *
                 * PARAMETERS:
                 *   - A.- POINTER TO A 4 BYTE VECTOR CONTAINING A FP NUMBER.
                 *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF
          -       ASCII CHARAC-
                 *     TERS IS DESIRED TO BE PLACED.
                 *   - NUMINT.- NUMBER OF CHARACTERS REPRESENTING THE DESIRED
          -      INTEGER
                 *     PORTION.
                 *   - NUMDEC.- SIMILAR TO NUMINT, BUT REPRESENTING THE DECIMA
          -      L PORTION
                 *     DESIRED.
                 *
                 * USAGE:
                 *   TYPED PROCEDURE. IF THE SIGN OF THE GIVEN FP NUMBER IS PO
          -      SITIVE, A
                 *   VALUE OF 0 IS RETURNED, OTHERWISE, A 1 IS RETURNED.
                 *
                 **************************************************************
```

```
                -   *************/
 151   1       FP$FORMAT: PROCEDURE (A,B,NUMINT,NUMDEC) BYTE PUBLIC;
 152   2          DCL (A,B) ADDRESS,
                       FP$NUM BASED A (4) BYTE,
                       STRING BASED B (16) BYTE,
                       BUFFER (28) BYTE,
                       (NUMINT,NUMDEC,NUM,SIGN,I,J,N,N1) BYTE;
 153   2          N1 = Ø;
 154   2          DO I = Ø TO NUMINT + NUMDEC;
 155   3             STRING (I) = 'Ø';
 156   3             END;
 157   2          CALL FLOAT$TO$ASCII(.FP$NUM,.BUFFER,.NUM);
 158   2          DO I = NUM TO 27;
 159   3             BUFFER(I) = 'Ø';
 16Ø   3             END;
 161   2          IF BUFFER(Ø) = ' '
                     THEN SIGN = Ø;
 163   2             ELSE SIGN = 1;
 164   2          I = 1;
 165   2          DO WHILE BUFFER(I) <> POINT;
 166   3             N1 = N1 + 1;
 167   3             I = I + 1;
 168   3             END;
 169   2          IF N1 > NUMINT
                  THEN DO;
 171   3                CALL CRT$PRINT$STRING(.('NUMBER TOO LARGE. $$'));
 172   3                HALT;
 173   3                END;
 174   2          I = 1;
 175   2          J = NUMINT - N1;
 176   2          N = NUMINT + NUMDEC;
 177   2          DO WHILE N > 0;
 178   3             IF BUFFER(I) = POINT THEN I = I + 1;
 180   3             STRING(J) = BUFFER(I);
 181   3             J = J + 1;
 182   3             I = I + 1;
 183   3             N = N - 1;
 184   3             END;
 185   2          IF (BUFFER(I) >= '5') AND (STRING(J-1) <> '9')      /* TO "
        -       ROUND". */
                     THEN STRING(J-1) = STRING(J-1) + 1;
 187   2          N = NUMINT + NUMDEC;
 188   2          STRING(N), STRING (N + 1) = '$';
 189   2          RETURN SIGN;
 190   2          END FP$FORMAT;


                /*************************************************************
        -        *************
                *
                * RANGE$FORMAT:
                *   THIS PROCEDURE IS USED TO OBTAIN IN ASCII CHARACTERS, THE
        -         VALUE OF A
                *   FP NUMBER REPRESENTING A RANGE. IF THE VALUE IS LESS THAN
        -         OR EQUAL TO 5
                *   MILES, THEN THE RANGE WILL BE GIVEN IN YARDS. OTHERWISE,I
        -        T WILL BE GIVEN
                *   IN MILES.
```

```
                 *
                 * PARAMETERS:
                 *   - A.- POINTER TO A 4 BYTE VECTOR REPRESENTING A FP NUMBER
               -  .
                 *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS
               -  DESIRED TO BE
                 *     PLACED.
                 *
                 ***********************************************************
               -  *****************/
 191   1         RANGE$FORMAT: PROCEDURE (A,B) PUBLIC;
 192   2            DCL (A,B) ADDRESS,
                        VALUE BASED A (4) BYTE,
                        STRING BASED B (6) BYTE,
                        RESULT (4) BYTE,
                        (TEMP, TWO) BYTE;
 193   2            TWO = 2;
 194   2            IF (TEMP:= FCMPR(A,.FP$5,.TWO))
                    THEN DO;
 196   3                 TEMP = FP$FORMAT(.VALUE,.STRING(0),3,1);
 197   3                 STRING(4) = STRING(3);
 198   3                 STRING(3) = POINT;
 199   3                 STRING(5) = 'M';
 200   3                 END;
 201   2            ELSE DO;
 202   3                 CALL FMUL(.VALUE,.FP$2000,.RESULT);
 203   3                 TEMP = FP$FORMAT(.RESULT,.STRING(0),5,0);
 204   3                 STRING(5) = 'Y';
 205   3                 END;
 206   2            END RANGE$FORMAT;
```

```
                  /*************************************************************
                -  *************
                  *
                  * LAT$LONG$FORMAT:
                  *   THIS PROCEDURE IS USED TO CONVERT A FLOATING POINT VALUE
                -  REPRESENTATION
                  *   OF LAT/LONG IN MINUTES, INTO A CORRESPONDING STRING OF CH
                -  ARACTERS.
                  *
                  * PARAMETERS:
                  *   - A.- POINTER TO A MEMORY LOCATION CONTAINING A FP VALUE.
                  *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF
                -   ASCII CHARAC-
                  *      TERS IS DESIRED TO BE PLACED.
                  *   - TYPE.- CAN HAVE ONE OF TWO VALUES:  0 ---> LATITUDE DES
                -  IRED.                                  1 ---> LONGITUDE DE
                  *
                -  SIRED.
                  *
                  *************************************************************
                -  *************/
 207   1          LAT$LONG$FORMAT: PROCEDURE (A,B,TYPE) PUBLIC;
 208   2             DCL (A,B) ADDRESS,
                         SIXTY ADDRESS DATA (060),
                         LAT$LONG BASED A (4) BYTE,
                         STRING BASED B (9) BYTE,
                         BUFFER(28) BYTE,
                            VALUE (4) BYTE,
                            TEMP (4) BYTE,
                            TEMP1 (4) BYTE,
                            REM (4) BYTE,
                            (NUM,I,SIGN,TYPE,J,TEST) BYTE;
 2Ø9   2               SIGN = Ø;
 21Ø   2               DO I = Ø TO 3;
 211   3                  VALUE(I) = LAT$LONG(I);
 212   3                  END;
 213   2               IF LAT$LONG(3) >= Ø8ØH
                       THEN DO;
 215   3                     SIGN = 1;
 216   3                     VALUE(3) = VALUE(3) XOR Ø8ØH;
 217   3                     END;
 218   2               CALL FIXSD(.VALUE,.TEMP);
 219   2               CALL EDIV(.TEMP,.SIXTY,.TEMP1,.REM);
 22Ø   2               CALL FLTDS(.TEMP1,.TEMP1);
 221   2               TEST = FP$FORMAT(.TEMP1,.STRING,3,Ø);
 222   2               CALL FLTDS(.REM,.REM);
 223   2               CALL FLTDS(.TEMP,.TEMP);
 224   2               CALL FSUB(.VALUE,.TEMP,.TEMP);
 225   2               CALL FADD(.TEMP,.REM,.TEMP);
 226   2               TEST = FP$FORMAT(.TEMP,.STRING(3),2,1);
 227   2               IF TYPE = Ø
                       THEN DO;
 229   3                     IF SIGN = 1 THEN STRING(6) = 'S';
 231   3                                 ELSE STRING(6) = 'N';
 232   3                     END;
 233   2               ELSE DO;
 234   3                     IF SIGN = 1 THEN STRING(6) = 'W';
 236   3                        ELSE STRING(6) = 'E';
 237   3                END;
 238   2            STRING(7),STRING(8) = '$';
 239   2            END LAT$LONG$FORMAT;


                   /*************************************************************
               -    ***********
                   *
                   * GET$TIME$ZONE:
                   *    THIS PROCEDURE IS USED TO OBTAIN A STRING OF ASCII CHARAC
               -    TERS FROM
                   *    THE CRT, REPRESENTING THE VALUE AND SIGN OF THE TIME ZONE
               -    .
                   *
                   * PARAMETERS:
                   *    - A.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS
               -     DESIRED TO
                   *       BE PLACED.
                   *
                   **************************************************************
               -    *************/
 240   1           GET$TIME$ZONE: PROCEDURE (A) PUBLIC;
 241   2               DCL A ADDRESS,
                           RESULT BASED A (5) BYTE,
                           (OK, SIGN, TEMP, VALUE) BYTE;
 242   2               OK = 0;
 243   2               DO WHILE OK = 0;
 244   3                   CALL CRT$PRINT$STRING(.('ENTER THE TIME ZONE VALUE AS R
                 -   EQUESTED:$$'));
 245   3               CALL SEND$CRLF;
 246   3               CALL CRT$PRINT$STRING(.('SIGN:  $$'));
 247   3               SIGN = CRT$READ;
 248   3               DO WHILE (SIGN <> '+') AND (SIGN <> '-');
 249   4                  CALL SEND$BEL;
 250   4                  SIGN = CRT$READ;
 251   4                  END;
 252   3               RESULT(0) = SIGN;
 253   3               CALL CRT$WRITE(SIGN);
 254   3               CALL SEND$CRLF;
 255   3               TEMP = 0;
 256   3               DO WHILE TEMP = 0;
 257   4                  CALL CRT$PRINT$STRING(.('VALUE:  $$'));
 258   4                  CALL PUT$NUMBER$BUFFER(2,.RESULT(1));
 259   4                  VALUE = ((RESULT(1) - 30H) * 10) + (RESULT(2) - 30H)
                 -   ;
 260   4                  IF VALUE > 12 THEN CALL PRINT$ERROR$MSG;
 262   4                  ELSE DO;
 263   5                       CALL CRT$WRITE(ETEOL);
 264   5                       TEMP = 1;
 265   5                       END;
 266   4                  END;
 267   3               CALL SEND$CRLF;
 268   3               OK = CHECK$INPUT;
 269   3               CALL CLEAR$LOW$SCREEN;
 270   3               END;
 271   2            RESULT(3), RESULT(4) = '$';
 272   2            END GET$TIME$ZONE;
```

```
                 /*************************************************************
           -      ***********
                 *
                 * GET$LAT:
                 *    PROCEDURE USED TO OBTAIN THE VALUES OF LATITUDE.
                 *
                 * PARAMETERS:
                 *    - A.- POINTER TO MEMORY LOCATION IN WHICH THE FP REPRESEN
           -      TATION
                 *       OF THE VALUE OF THE LATITUDE IN MINUTES IS DESIRED TO B
           -      E PLACED.
                 *
                 **************************************************************
           -      ************/
 273   1         GET$LAT: PROCEDURE (A) PUBLIC;
 274   2            DCL A ADDRESS,
                         RESULT BASED A (4) BYTE,
                         OP1 (4) BYTE,
                         OP2 (4) BYTE,
                         (SIGN,OK,TEST,FIVE) BYTE;
 275   2            FIVE = 5;
 276   2            OK = 0;
 277   2            DO WHILE OK = 0;
 278   3               CALL CRT$PRINT$STRING(.('ENTER THE LATITUDE VALUE AS RE
           -      QUESTED:$$'));
 279   3               CALL SEND$CRLF;
 280   3               CALL GET$DEGREES(2,.OP1);
 281   3               CALL SEND$CRLF;
```

```
            CALL GET$MINUTES(.OP2);
            CALL SEND$CRLF;
            CALL FADD(.OP1,.OP2,.RESULT);
            SIGN = GET$SIGN('N','S');
            IF SIGN = 0
            THEN DO;
               IF (TEST:= FZTST(.RESULT,.FIVE)) THEN
                  RESULT(3) = RESUIT(3) XOR 080H;
               CALL CRT$PRINT$STRING(.('OUTH.$$'));
               END;
            ELSE CALL CRT$PRINT$STRING(.('ORTH.$$'));
            CALL SEND$CRLF;
            OK = CHECK$INPUT;
            CALL CLEAR$LOW$SCREEN;
            END;
         END GET$LAT;


        /************************************************************
   -     ***************
         *
         * GET$LONG:
         *   PROCEDURE USED TO OBTAIN THE FP REPRESENTATION OF THE LON
   -     GITUDE IN
         *   MINUTES.
         *
         * PARAMETERS:
         *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF
   -     THE
```

```
                     *         LONGITUDE IN MINUTES IS DESIRED TO BE STORED.
                     *
                     *************************************************************
                   -  **************/
 298    1            GET$LONG: PROCEDURE (A) PUBLIC;
 299    2               DCL A ADDRESS,
                             RESULT BASED A (4) BYTE,
                             OP1 (4) BYTE,
                             OP2 (4) BYTE,
                             (SIGN,OK,TEST,FIVE) BYTE;
 300    2               FIVE = 5;
 301    2               OK = 0;
 302    2               DO WHILE OK = 0;
 303    3                  CALL CRT$PRINT$STRING(.('ENTER THE LONGITUDE VALUE AS R
                   -  EQUESTED:$$'));
 304    3                  CALL SEND$CRLF;
 305    3                  CALL GET$DEGREES(3,.OP1);
 306    3                  CALL SEND$CRLF;
 307    3                  CALL GET$MINUTES(.OP2);
 308    3                  CALL SEND$CRLF;
 309    3                  CALL FADD(.OP1,.OP2,.RESULT);
 310    3                  SIGN = GET$SIGN('E','W');
 311    3                  IF SIGN = 0
                           THEN DO;
 313    4                     CALL CRT$PRINT$STRING(.('EST$$'));
 314    4                     IF (TEST:= F2TST(.RESULT,.FIVE)) THEN
 315    4                        RESULT(3) = RESULT(3) XOR 080H;
 316    4                     END;
 317    3                  ELSE CALL CRT$PRINT$STRING(.('AST$$'));
 318    3                  CALL SEND$CRLF;
 319   3             OK = CHECK$INPUT;
 320   3             CALL CLEAR$LOW$SCREEN;
 321   3             END;
 322   2          END GET$LONG;


            /*************************************************************
       -     ************
            *
            * GET$COURSE$BRG:
            *    THIS PROCEDURE IS USED TO OBTAIN THE VALUE OF COURSE OR B
       -     EARING FROM
            *    THE CRT.
            *
            * PARAMETERS:
            *    - TYPE.- HAS TWO POSSIBLE VALUES: IF 0 ---> GET COURSE.
            *                                      IF 1 ---> GET BEARING.
            *
            *    - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING
       -     POINT REPRE-
            *         SENTATION OF THE VALUE OF THE COURSE OR BEARING, IN DEG
       -     REES, IS DE-
            *         SIRED TO BE PLACED.
            *
            **************************************************************
       -     *************/
 323   1    GET$COURSE$BRG: PROCEDURE (TYPE,A) PUBLIC;
 324   2       DCL A ADDRESS,
                   RESULT BASED A (4) BYTE,
                   BUFFER(7) BYTE,
                         (TEMP, OK, TYPE) BYTE;
 325    2            OK = 0;
 326    2            BUFFER(0) = 4;
 327    2            BUFFER(1) = 3;
 328    2            BUFFER(6) = 0;
 329    2            DO  WHILE OK = 0;
 330    3               CALL CRT$PRINT$STRING(.('ENTER THE $$'));
 331    3               IF TYPE = 0
                           THEN CALL CRT$PRINT$STRING(.('COURSE $$'));
 333    3                  ELSE CALL CRT$PRINT$STRING(.('BEARING $$'));
 334    3               CALL CRT$PRINT$STRING(.('VALUE AS REQUESTED:$$'));
 335    3               CALL SEND$CRLF;
 336    3               TEMP = 0;
 337    3               DO WHILE TEMP = 0;
 338    4                  CALL CRT$PRINT$STRING(.('DEGREES:  $$'));
 339    4                  CALL PUT$NUMBER$BUFFER(3,.BUFFER(2));
 340    4                  CALL CRT$WRITE(POINT);
 341    4                  CALL PUT$NUMBER$BUFFER(1,.BUFFER(5));
 342    4                  CALL ASCII$TO$FLOAT(.BUFFER,7,.RESULT);
 343    4                  TEMP = CHECK$FP$VALUE(.RESULT,.FP$360);
 344    4                  END;
 345    3               CALL SEND$CRLF;
 346    3               OK = CHECK$INPUT;
 347    3               CALL CLEAR$LOW$SCREEN;
 348    3               END;
 349    2            END GET$COURSE$BRG;
```

```
                    /*************************************************************
                  -  **************
                    *
                    * GET$SPEED:
                    *   THIS PROCEDURE IS USED TO OBTAIN THE SPEED VALUE FROM THE
                  -   CRT.
                    *
                    * PARAMETERS:
                    *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING
                  -  POINT REPRESEN-
                    *     TATION OF THE VALUE OF THE SPEED IN KNOTS,IS DESIRED TO
                  -   BE PLACED.
                    *
                    *************************************************************
                  -  **************/
 350    1           GET$SPEED: PROCEDURE (A) PUBLIC;
 351    2             DCL A ADDRESS,
                          RESULT BASED A (4) BYTE,
                          BUFFER(6) BYTE,
                          (TEMP,OK) BYTE;
 352    2             OK = 0;
 353    2             BUFFER(0) = 3;
 354    2             BUFFER(1) = 2;
 355    2             BUFFER(5) = 0;
 356    2             DO WHILE OK = 0;
 357    3                CALL CRT$PRINT$STRING(.('ENTER THE SPEED VALUE AS REQUE
                  -  STED:$$'));
 358    3                CALL SEND$CRLF;
 359    3                TEMP = 0;
 360    3                DO WHILE TEMP = 0;
```

```
361   4              CALL CRT$PRINT$STRING(.('KNOTS:  $$'));
362   4              CALL PUT$NUMBER$BUFFER(2,.BUFFER(2));
363   4              CALL CRT$WRITE(POINT);
364   4              CALL PUT$NUMBER$BUFFER(1,.BUFFER(4));
365   4              CALL ASCII$TO$FLOAT(.BUFFER,6,.RESULT);
366   4              TEMP = CHECK$FP$VALUE(.RESULT,.FP$99$9);
367   4              END;
368   3           CALL SEND$CRLF;
369   3           OK = CHECK$INPUT;
370   3           CALL CLEAR$LOW$SCREEN;
371   3           END;
372   2        END GET$SPEED;


            /*****************************************************************
-            *************
             *
             * GET$RANGE:
             *    THIS PROCEDURE IS USED TO OBTAIN THE FLOATING POINT REPRE
-            SENTATION OF
             *    A RANGE VALUE OBTAINED FROM THE CRT.
             *
             * PARAMETERS:
             *    - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING
-            POINT REPRESEN-
             *       TATION OF THE VALUE OF A RANGE, IS DESRED TO BE PLACED.
             *
             * USAGE:
             *    ALTHOUGH THIS PROCEDURE WILL ACCEPT EITHER YARDS OR MILES
            -   AS UNITS OF
                *   RANGE, THE RESULT WILL BE GIVEN IN TERMS OF MILES ONLY.
                *
                *************************************************************
            -    **************/
 373    1       GET$RANGE: PROCEDURE (A) PUBLIC;
 374    2          DCL A ADDRESS,
                       RESULT BASED A (4) BYTE,
                       BUFFER (8) BYTE,
                       (TEMP, OK, CHAR, UNITS) BYTE;
 375    2          OK = Ø;
 376    2          DO WHILE OK = Ø;
 377    3             CALL CRT$PRINT$STRING(.('ENTER THE RANGE VALUE AS REQUE
            -    STED:$$'));
 378    3             CALL SEND$CRLF;
 379    3             CALL CRT$PRINT$STRING(.('ENTER THE UNITS TO BE USED:  (
            -    M/Y)  $$'));
 38Ø    3             CHAR = CRT$READ;
 381    3             DO WHILE (CHAR <> 'Y') AND (CHAR <> 'M')       /* UPPER
            -    CASE CHARACTERS. */
                          AND (CHAR <> 'y') AND (CHAR <> 'm');      /* LOWER
            -    CASE CHARACTERS. */
 382    4                CALL SEND$BEL;
 383    4                CHAR = CRT$READ;
 384    4                END;
 385    3             IF CHAR >= 61H THEN CHAR = CHAR - Ø2ØH;
 387    3             IF CHAR = 'Y' THEN CALL CRT$PRINT$STRING(.('YARDS.$$'))
            -    ;
 389    3                                 ELSE CALL CRT$PRINT$STRING(.('MILES.$$'
            -    ));
```

```
390    3            CALL SEND$CRLF;
391    3            TEMP = 0;
392    3            DO WHILE TEMP = 0;
393    4               IF CHAR = 'Y'
                       THEN DO;
395    5                  CALL CRT$PRINT$STRING(.('YARDS:  $$'));
396    5                  CALL PUT$NUMBER$BUFFER(6,.BUFFER(2));
397    5                  BUFFER(0), BUFFER(1) = 6;
398    5                  BUFFER(8) = 0;
399    5                  CALL ASCII$TO$FLOAT(.BUFFER,9,.RESULT);
400    5                  TEMP = CHECK$FP$VALUE(.RESULT,.FP$202500);
401    5                  CALL FDIV(.RESULT,.FP$2000,.RESULT);
402    5                  END;
403    4               ELSE DO;
404    5                  CALL CRT$PRINT$STRING(.('MILES:  $$'));
405    5                  CALL PUT$NUMBER$BUFFER(3,.BUFFER(2));
406    5                  CALL CRT$WRITE(POINT);
407    5                  CALL PUT$NUMBER$BUFFER(1,.BUFFER(5));
408    5                  BUFFER(0) = 4;
409    5                  BUFFER(1) = 3;
410    5                  BUFFER(6) = 0;
411    5                  CALL ASCII$TO$FLOAT(.BUFFER,7,.RESULT);
412    5                  TEMP = CHECK$FP$VALUE(.RESULT,.FP$100);
413    5                  END;
414    4               END;
415    3            CALL SEND$CRLF;
416    3            OK = CHECK$INPUT;
417    3            CALL CLEAR$LOW$SCREEN;
418    3            END;
419    2         END GET$RANGE;
```

```
                /**************************************************************
            -    ***************
                *
                * GET$DESIG:
                *    THIS PROCEDURE IS USED TO OBTAIN THE DESIGNATION OF THE C
            -    ONTACT FROM
                *    THE CRT.
                *
                * USAGE:
                *    TYPED PROCEDURE. RETURNS A HASHED VALUE TO BE USED INTERN
            -    ALLY ACCORDING
                *    TO THE FOLLOWING RULE:
                *              HASH = CHAR*100 + CHAR1.
                *
                **************************************************************
            -    ***************/
 420   1        GET$DESIG: PROCEDURE ADDRESS PUBLIC;
 421   2           DCL HASH ADDRESS,
                       BUFFER (2) BYTE,
                       (CHAR, OK, CHAR1) BYTE;
 422   2           OK = 0;
 423   2           DO WHILE OK = 0;
 424   3              CALL CRT$PRINT$STRING(.('DESIG:  $$'));
 425   3           L: CHAR = CRT$READ;
 426   3              DO WHILE ((CHAR < 41H) AND (CHAR <> 20H)) OR
                              ((CHAR > 5AH) AND (CHAR < 61H)) OR (CHAR > 7AH
            -    );
 427   4              CALL SEND$BEL;
 428   4              CHAR = CRT$READ;
 429   4              END;
 430   3            IF CHAR >= 61H THEN CHAR = CHAR - 20H;
 432   3            CALL CRT$WRITE(CHAR);
 433   3            CHAR1 = CRT$READ;
 434   3            DO WHILE ((CHAR1 < 41H) OR ((CHAR1 > 5AH) AND
                          (CHAR1 < 61H)) OR (CHAR1 > 7AH)) AND (CHAR1 <>
     -       07FH);
 435   4              CALL SEND$BEL;
 436   4              CHAR1 = CRT$READ;
 437   4              END;
 438   3            IF CHAR1 = 07FH THEN DO;
 440   4              CALL SEND$BS;
 441   4              GO TO L;
 442   4              END;
 443   3            IF CHAR1 >= 61H THEN CHAR1 = CHAR1 - 20H;
 445   3            CALL CRT$WRITE(CHAR1);
 446   3            CALL SEND$CRLF;
 447   3            OK = CHECK$INPUT;
 448   3            CALL CLEAR$LOW$SCREEN;
 449   3            END;
 450   2          HASH = CHAR*100 + CHAR1;
 451   2          RETURN HASH;
 452   2          END GET$DESIG;


             /**************************************************************
     -        ***************
```

```
                 *
                 * GET$TYPE:
                 *   THIS PROCEDURE IS USED TO OBTAIN THE TYPE OF THE CONTACT
               -  FROM
                 *   THE CRT.
                 *
                 * USAGE:
                 *   TYPED PROCEDURE. RETURNS THE FOLLOWING VALUES :
                 *                     0 ---> SURFACE.
                 *                     1 ---> SUB-SURFACE.
                        2 ---> AIR.
                 *
                 ************************************************************
               -  ***************/
 453   1         GET$TYPE: PROCEDURE BYTE PUBLIC;
 454   2            DCL (TYPE, CHAR, OK) BYTE;
 455   2            OK = 0;
 456   2            DO WHILE OK = 0;
 457   3               CALL CRT$PRINT$STRING(.('ENTER THE CONNTACT TYPE: SURFA
               -  CE = 2 / SUBSURFACE = 3 / AIR = 1  (1/2/3)  $$'));
 458   3               CALL GET$STRING(.CHAR,1);
 459   3               IF (CHAR <> '1') AND (CHAR <> '2') AND (CHAR <> '3')
                          THEN CALL PRINT$ERROR$MSG;
 461   3              ELSE DO;
 462   4                  CALL CRT$WRITE(ETEOL);
 463   4                  IF CHAR = '1'
                         THEN DO;
 465   5                       CALL CRT$PRINT$STRING(.(' - AIR. $$'));
 466   5                       TYPE = 2;
 467   5                       END;
 468   4               ELSE DO;
 469   5                  IF CHAR = '2'
                         THEN DO;
 471   6                     CALL CRT$PRINT$STRING(.(' - SURFACE.  $$'))
            -   ;
 472   6                     TYPE = 0;
 473   6                     END;
 474   5                  ELSE DO;
 475   6                     CALL CRT$PRINT$STRING(.(' - SUB-SURFACE.
            -   $$'));
 476   6                     TYPE = 1;
 477   6                     END;
 478   5                  END;
 479   4               END;
 480   3            CALL SEND$CRLF;
 481   3            OK = CHECK$INPUT;
 482   3            CALL CLEAR$LOW$SCREEN;
 483   3            END;
 484   2          RETURN TYPE;
 485   2          END GET$TYPE;


                /***********************************************************
            -    ****************
                *
                * GET$KIND:
                *   THIS PROCEDURE IS USED TO OBTAIN THE CONTACTS KIND FROM T
            -    HE CRT.
                *
```

```
                  * USAGE:
                  *   TYPED PROCEDURE. RETURNS THE FOLLOWING VALUES:
                  *                        0 ---> FRIEND.
                  *                        1 ---> HOSTILE.
                  *                        2 ---> UNKNOWN.
                  *
                  **************************************************************
                -  ****************/
 486   1          GET$KIND: PROCEDURE BYTE PUBLIC;
 487   2            DCL (KIND, TEMP, CHAR, OK) BYTE;
 488   2            OK = 0;
 489   2            DO WHILE OK = 0;
 490   3              TEMP = 0;
 491   3              DO WHILE TEMP = 0;
 492   4                CALL CRT$PRINT$STRING(.('ENTER THE CONTACT CLASS: (F
                -  /H/U)  $$'));
 493   4                CALL GET$STRING(.CHAR,1);
 494   4                IF (CHAR <> 'F') AND (CHAR <> 'H') AND (CHAR <> 'U')
                            THEN CALL PRINT$ERROR$MSG;
 496   4                    ELSE DO;
 497   5                        CALL CRT$WRITE(ETBOL);
 498   5                        TEMP = 1;
 499   5                        IF CHAR = 'F'
                                    THEN DO;
 501   6                                CALL CRT$PRINT$STRING(.('RIEND.$$'))
                -  ;
 502   6                                KIND = 0;
 503   6                                END;
 504   5                            ELSE DO;
 505   6                                IF CHAR = 'H'
```

```
                                             THEN DO;
                                                 CALL CRT$PRINT$STRING(.('OST
 507   7
                 -   ILE.$$'));
                                                 KIND = 1;
 508   7                                         END;
 509   7                                     ELSE DO;
 510   6                                         CALL CRT$PRINT$STRING(.('NEN
 511   7
                 -   OWN.$$'));
 512   7                                         KIND = 2;
 513   7                                         END;
 514   6                                     END;
 515   5                               END;
 516   4                         END;
 517   3                      CALL SEND$CRLF;
 518   3                      OK = CHECK$INPUT;
 519   3                      CALL CLEAR$LOW$SCREEN;
 520   3                      END;
 521   2                  RETURN KIND;
 522   2                  END GET$KIND;


                     /*************************************************************
                 -    ***************
                     *
                     * GET$SCALE:
                     *    THIS PROCEDURE IS USED TO OBTAIN A NUMBER REPRESENTING TH
                 -    E SCALE DESIRED
                     *    TO BE USED IN THE PLOTTING AT THE PLASMA DISPLAY.
                     *
```

```
             * PARAMETERS:
             *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING
           -  POINT REPRESEN-
             *     TATION OF THE SCALE (MILES/INCH) IS DESIRED TO BE PLACE
           -  D.
             *
             *****************************************************************
           -  ****************/
      1      GET$SCALE: PROCEDURE (A) PUBLIC;
      2         DCL A ADDRESS,
                    SCALE BASED A (4) BYTE,
                    BUFFER (7) BYTE,
                    (TEMP,TEMP1,OK) BYTE;
      2         OK = 0;
      2         DO WHILE OK = 0;
      3            CALL CRT$PRINT$STRING(.('ENTER THE SCALE AS REQUESTED:$
           -  $'));
      3            CALL SEND$CRLF;
      3            TEMP, TEMP1 = 0;
      3            DO WHILE (TEMP = 0) OR (TEMP1 = 0);
      4               CALL CRT$PRINT$STRING(.('MILES PER INCH: $$'));
      4               CALL PUT$NUMBER$BUFFER(2,.BUFFER(2));
      4               CALL CRT$WRITE(POINT);
      4               CALL PUT$NUMBER$BUFFER(2,.BUFFER(4));
      4               BUFFER(0) = 4;
      4               BUFFER(1) = 2;
      4               BUFFER(6) = 0;
      4               CALL ASCII$TO$FLOAT(.BUFFER,7,.SCALE);
      4               TEMP = CHECK$FP$VALUE(.SCALE,.FP$25);
      4               IF TEMP <> 0 THEN
```

```
 541   4                  TEMP1 = CHECK$FP$VALUE(.FP$0$25,.SCALE);
 542   4                END;
 543   3             CALL SEND$CRLF;
 544   3             OK = CHECK$INPUT;
 545   3             CALL CLEAR$LOW$SCREEN;
 546   3             END;
 547   2          END GET$SCALE;

 548   1       END COMMANDS;


MODULE INFORMATION:

     CODE AREA SIZE     = 0F67H   3943D
     VARIABLE AREA SIZE = 00E6H    230D
     MAXIMUM STACK SIZE = 0008H      8D
     955 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
                 /************************************************************
               -  **************
                 *
                 * ACTUAL$TIME:
                 *   PROCEDURE USED TO PUT INTO THE VECTOR 'TIME$BUFFER', THE
               -  ACTUAL
                 *   TIME IN A STRING FORM.
                 *
                 ************************************************************
               -  *************/
 129   1         ACTUAL$TIME: PROCEDURE PUBLIC;
 130   2            TIME$BUFFER(0) = HOURS/10 + 30H;
 131   2            TIME$BUFFER(1) = HOURS MOD 10 + 30H;
 132   2            TIME$BUFFER(2) = MINUTES/10 + 30H;
 133   2            TIME$BUFFER(3) = MINUTES MOD 10 + 30H;
 134   2            TIME$BUFFER(4) = SECONDS/10 + 30H;
 135   2            TIME$BUFFER(5) = SECONDS MOD 10 + 30H;
 136   2            END ACTUAL$TIME;


 137   1         END TIME;



MODULE INFORMATION:

     CODE AREA SIZE     = 0255H    597D
     VARIABLE AREA SIZE = 0011H     17D
     MAXIMUM STACK SIZE = 0008H      8D
     225 LINES READ
```

```
                  /**************************************************************
              -    *****
                  *
                  * INITIATE$CLOCK:
                  *    PROCEDURE USED TO START THE SIMULATED REAL TIME CLOCK.
                  *
                  ***************************************************************
              -    ******/
 121   1          INITIATE$CLOCK: PROCEDURE PUBLIC;
 122   2            MILI$SEC,DUMMY$SEC = 00;
                  /*
                  *                               ***   WARNING   ***
                  *   THE FOLLOWING STATEMENT SHOULD BE REMOVED IF THIS MODULE
              -    IS TO BE
                  *   EXECUTED WITHOUT ISIS. THE CLOCK PROCEDURE SHOULD ALSO BE
              -     RECOMPI-
                  *   LED AS PROCEDURE INTERRUPT 1.
                  *
                  */
 123   2             CALL MOVE (3,038H,008H);
 124   2             OUTPUT(0FDH) = 12H;
 125   2             OUTPUT(0FCH) = 00H;
 126   2             ENABLE;
 127   2             OUTPUT(0FFH) = 00H;
 128   2             END INITIATE$CLOCK;
```

```
  96   5                    END;
  97   4                 ELSE DO;
  98   5                    CALL CRT$WRITE(17H);         /*  ERASE TO END OF L
            -   INE  */
  99   5                    CALL SEND$CRLF;
 100   5                    END;
 101   4                 END;
 102   3              DO WHILE SECONDS >= 60;
 103   4                 CALL CRT$PRINT$STRING(.('SECONDS: $$'));
 104   4                 SECONDS = GET$BYTE(2);
 105   4                 IF SECONDS >= 60
                         THEN DO;
 107   5                    CALL CRT$PRINT$STRING(.MSG);
 108   5                    CALL SEND$BEL;
 109   5                    CALL SEND$CR;
 110   5                    CALL SEND$SUB;
 111   5                    END;
 112   4                 ELSE DO;
 113   5                    CALL CRT$WRITE(17H);          /*  ERASE TO END OF
            -   LINE  */
 114   5                    CALL SEND$CRLF;
 115   5                    END;
 116   4                 END;
 117   3              OK = CHECK$INPUT;
 118   3              CALL CLEAR$LOW$SCREEN;
 119   3              END;

 120   2           END INITIATE$TIME;
 68   2          DO WHILE OK = 0;
 69   3             HOURS, MINUTES, SECONDS = 0FFH;
 70   3             CALL CRT$PRINT$STRING(.('INPUT THE TIME AS REQUESTED.$$
           -    '));
 71   3             CALL SEND$CRLF;
 72   3             DO WHILE HOURS >= 24;
 73   4                CALL CRT$PRINT$STRING(.('HOURS: $$'));
 74   4                HOURS = GET$BYTE(2);
 75   4                IF HOURS >= 24
                       THEN DO;
 77   5                   CALL CRT$PRINT$STRING(.MSG);
 78   5                   CALL SEND$BEL;
 79   5                   CALL SEND$CR;
 80   5                   CALL SEND$SUB;
 81   5                   END;
 82   4                ELSE DO;
 83   5                   CALL CRT$WRITE(17H);        /*  ERASE TO END OF LI
           -    NE  */
 84   5                   CALL SEND$CRLF;
 85   5                   END;
 86   4                END;
 87   3             DO WHILE MINUTES >= 60;
 88   4                CALL CRT$PRINT$STRING(.('MINUTES: $$'));
 89   4                MINUTES = GET$BYTE(2);
 90   4                IF MINUTES >= 60
                       THEN DO;
 92   5                   CALL CRT$PRINT$STRING(.MSG);
 93   5                   CALL SEND$BEL;
 94   5                   CALL SEND$CR;
 95   5                   CALL SEND$SUB;
```

```
                  /*   RESTORE CURRENT OPERATING LEVEL.   */
  58   2             OUTPUT(0FDH) = 020H;
                  /*   SET THE MDS REAL TIME CLOCK.   */
  59   2             TEMP = INPUT(0FFH);
  60   2             TEMP = INPUT(0FFH);
  61   2             OUTPUT(0FFH) = 00H;
                  /*   THE RETURN STATEMENT WILL ENABLE INTERRUPTS AUTOMATICALL
                -  Y.   */
  62   2             RETURN;
  63   2             END CLOCK;


                  /****************************************************************
                -  ****************
                  *
                  * INITIATE$TIME:
                  *   PROCEDURE USED DURING SYSTEM INITIALIZATION. USED TO SET
                -  THE SIMULATED
                  *   REAL TIME CLOCK TO AN SPECIFIED HOUR.
                  *
                  * USAGE:
                  *   UTILIZES THE GLOBAL VARIABLES HOURS,MINUTES AND SECONDS.
                  *
                  *****************************************************************
                -  **************/
  64   1          INITIATE$TIME: PROCEDURE PUBLIC;
  65   2             DECLARE OK BYTE;
  66   2             DECLARE MSG(*) BYTE DATA (' ***  BAD FORMAT.  ***$$');
  67   2             OK = 0;
```

```
 26   1       CLOCK: PROCEDURE INTERRUPT 7;
 27   2          DECLARE TEMP BYTE;
              /*   TO RESET THE MDS REAL TIME CLOCK.   */
 28   2          OUTPUT(ØFFH) = Ø3H;
 29   2          MILI$SEC = MILI$SEC + 1;
 3Ø   2          IF MILI$SEC = 128 THEN DO;
 32   3             MILI$SEC = Ø;
 33   3             DUMMY$SEC = DUMMY$SEC + 1;
 34   3             END;
 35   2          IF DUMMY$SEC = Ø8H THEN DO;
 37   3             MILI$SEC,DUMMY$SEC = Ø;
 38   3             SEC$TIME = ØFFH;          /*   BOOLEAN VARIABLE. A SECOND
            -    HAS ELAPSED. */
 39   3             SECONDS = SECONDS + 1;
 4Ø   3             TIME$STEP = TIME$STEP + 1;
 41   3             IF SECONDS = 6Ø THEN DO;
 43   4                SECONDS = ØØ;
 44   4                MINUTES = MINUTES + 1;
 45   4                IF MINUTES = 6Ø THEN DO;
 47   5                   MINUTES = ØØ;
 48   5                   HOURS = HOURS + 1;
 49   5                   IF HOURS = 24 THEN DO;
 51   6                      HOURS = ØØ;
 52   6                      DAY = 1;
 53   6                      END;
 54   5                   END;
 55   4                END;
 56   3             END;
              /*   DISABLE INTERRUPTS.   */
 57   2          DISABLE;
```

```
            * CLOCK:
            *   THIS PROCEDURE IS OF THE TYPE INTERRUPT. IT IS USED TO MA
      -     INTAIN A REAL
            *   TIME HOUR, ONCE INITIATED. IT USES THE MDS REAL TIME CLOC
      -     K.
            *
            * USAGE:
            *   THIS PROCEDURE IS CALLED EACH TIME AN INTERRUPT FROM THE
      -     REAL TIME CLOCK
            *   IN THE MDS SYSTEM, IS PRODUCED.
            *
            *                          ***   WARNING   ***
            * SINCE THE DEVELOPMENT OF THIS PROCEDURE WAS DONE UNDER ISIS
      -      , THE PROCEDURE
            * HAD TO BE DECLARED AS INTERRUPT 7, SINCE ISIS DOES NOT ALLO
      -     W FOR INTERRUPTS
            * LESS THAN OR EQUAL TO 2. THE INTERRUPT FROM THE REAL TIME C
      -     LOCK IS OF LEVEL
            * 1, AND THEREFORE, A 'CALL MOVE' INSTRUCTION HAD TO BE IMPLE
      -     MENTED, IN
            * ORDER TO MOVE THE CODE IN INT 7 TO INT 1 AND PASS OVER THIS
      -       ISIS INCONVENIEN-
            * CE. IF A COPY OF THIS PROGRAM IS TO BE EXECUTED WITHOUT ISI
      -     S, THEN THIS PRO-
            * CEDURE MUST BE RECOMPILED AS PROCEDURE INTERRUPT 1, AND THE
      -       'CALL MOVE' STA-
            * TEMENT IN THE INITIATE$CLOCK PROCEDURE, REMOVED.
            *
            *****************************************************************
      -      ****************/
```

```
 14   1      SEND$CRLF:
               PROCEDURE EXTERNAL;
 15   2        END;

 16   1      SEND$SUB:
               PROCEDURE EXTERNAL;
 17   2        END;

 18   1      GET$BYTE:
               PROCEDURE (A) BYTE EXTERNAL;
 19   2        DECLARE A BYTE; END;

 21   1      CHECK$INPUT:
               PROCEDURE BYTE EXTERNAL;
 22   2        END;

 23   1      CLEAR$LOW$SCREEN:
               PROCEDURE EXTERNAL;
 24   2        END;

 25   1      DECLARE (MILI$SEC,DUMMY$SEC,SECONDS,MINUTES,HOURS,DAY,SEC$TIM
          -   E) BYTE PUBLIC,
                    TIME$STEP ADDRESS PUBLIC,
                    TIME$BUFFER(6) BYTE PUBLIC;

             /*****************************************************************
          -   ****************
             *
```

```
PL/M-80 COMPILER                                          20  MAR 8  PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE TIME
OBJECT MODULE PLACED IN TIME.OBJ
COMPILER INVOKED BY:  :F1:PLM80 TIME.SRC PAGELENGTH(53) PAGEWIDTH(75) DATE(
                    -20  MAR 81)


   1          TIME: DO;


   2   1      CRT$WRITE:
                PROCEDURE (A) EXTERNAL;
   3   2        DECLARE A BYTE; END;

   5   1      CRT$PRINT$STRING:
                PROCEDURE (A) EXTERNAL;
   6   2        DECLARE A ADDRESS; END;

   8   1      ECHO$CRT:
                PROCEDURE BYTE EXTERNAL;
   9   2        END;

  10   1      SEND$BEL:
                PROCEDURE EXTERNAL;
  11   2        END;

  12   1      SEND$CR:
                PROCEDURE EXTERNAL;
  13   2        END;
```

```
                                                      20 MAR 81   PAGE    1
PL/M-80 COMPILER

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE DISPLAYCMDS
OBJECT MODULE PLACED IN DISP.OBJ
COMPILER INVOKED BY:   :F1:PLM80 DISP.SRC PAGELENGTH(33) PAGEWIDTH(75) DATE(
                  -20 MAR 81)



   1            DISPLAY$CMDS: DO;


                $NOLIST

 252    1       GET$CPA:
                   PROCEDURE (A,B) BYTE EXTERNAL;
 253    2          DCL A BYTE, B ADDRESS; END;

 255    1       CONV$CONTACT$TIME:
                   PROCEDURE (A, B) EXTERNAL;
 256    2          DCL (A, B) ADDRESS; END;

 258    1       DCL SAFE$RNG (4) BYTE EXTERNAL;

 259    1       DCL
                    TITLE$0 (*) BYTE DATA                      COORDINATE GRID OR
                            ('
            -    IGIN$$'),
                    TITLE$1 (*) BYTE DATA                        GRAPHICS SCALE
                            ('
           -   $$'),
                  TITLE$2 (*) BYTE DATA
                          ('                              OWN SHIP INFORMAT
           -   ION$$'),
                  TITLE$3 (*) BYTE DATA
                          ('                              CONTACT INFORMATI
           -   ON$$'),
                  TITLE$4 (*) BYTE DATA
                          ('                               SYSTEM INFORMATI
           -   ON.$$'),
                  TITLE$5 (*) BYTE DATA
                          ('                         CURRENT SAFE C.P.A. R
           -   ANGE $$'),
                  TITLE$6 (*) BYTE DATA
                          ('                              WIND INFORMATIO
           -   N$$'),
                  TITLE$7 (*) BYTE DATA
                          ('                       CURRENT TIME BETWEEN U
           -   PDATES.$$');

 260   1       DCL
                  MSG$00 (*) BYTE DATA ('POSITIONAL DATA:$$'),
                  MSG$01 (*) BYTE DATA ('TACTICAL DATA AT $$'),
                  MSG$02 (*) BYTE DATA ('C.P.A. DATA:$$'),
                  MSG$03 (*) BYTE DATA ('GENERAL DATA:$$');

 261   1       DCL
                  PLUS$SIGN  LIT '02BH',
                  MINUS$SIGN LIT '02DH',
              COLON LIT '03AH',
              POINT LIT '02EH',
              BLANK LIT '020H';

       /*************************************************************
-       ***************
       *
       * CONV$LAT$LONG:
       *   THIS PROCEDURE IS USED TO CONVERT GIVEN 'X,Y' COORDINATES
-       , INTO LATITUDE
       *   AND LONGITUDE VALUES.
       *
       * PARAMETERS:
       *    - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRES
-       ENTATION OF 'X'
       *      IS LOCATED.
       *    - B.- POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRES
-       ENTATION OF 'Y'
       *      IS LOCATED.
       *    - C.- POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF
-       THE LATITUDE, IN
       *      MINUTES, IS DESIRED TO BE PLACED.
       *    - D.- POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF
-       THE LONGITUDE, IN
       *      MINUTES, IS DESIRED TO BE PLACED.
       *
       *************************************************************
-       ***************/
```

```
 262   1       CONV$LAT$LONG: PROCEDURE (A,B,C,D) PUBLIC;
 263   2          DCL (A,B,C,D) ADDRESS,
                     X BASED A (4) BYTE,
                     Y BASED B (4) BYTE,
                     LAT BASED C (4) BYTE,
                     LONG BASED D (4) BYTE,
                     MEAN$LAT (4) BYTE,
                     COS$MEAN$LAT (4) BYTE,
                     SIN$MEAN$LAT (4) BYTE,
                     FP$2 (4) BYTE DATA (00H, 00H, 80H, 40H);
 264   2          CALL FADD(.Y, .SYSTEM.LAT, .LAT);
 265   2          CALL FADD(.SYSTEM.LAT, .LAT, .MEAN$LAT);
 266   2          CALL FDIV(.MEAN$LAT, .FP$2, .MEAN$LAT);
 267   2          CALL CONV$MIN$RAD(.MEAN$LAT, .MEAN$LAT);
 268   2          CALL COS$SIN(.MEAN$LAT, .COS$MEAN$LAT, .SIN$MEAN$LAT);
 269   2          CALL FDIV(.X, .COS$MEAN$LAT, .LONG);
 270   2          CALL FADD(.LONG, .SYSTEM.LONG, .LONG);
 271   2          END CONV$LAT$LONG;


           /*********************************************************
  -         ****************
           *
           * DISPLAY$DESIG:
           *   THIS PROCEDURE IS USED TO DISPLAY THE DESIG CHARACTERS.
           *
           * PARAMETERS:
           *   - DESIG.- ADDRESS VALUE REPRESENTING THE 'HASHED' VALUE O
  -         F THE DESIGNATION
```

```
                *    DESIRED TO BE DISPLAYED.
                *
                ****************************************************************
              -  *****************/
272   1         DISPLAY$DESIG: PROCEDURE(DESIG) PUBLIC;
273   2            DCL DESIG ADDRESS,
                       CHAR(4) BYTE;
274   2            DCL D (10) BYTE DATA ('DESIG:  $$');
275   2            CALL CRT$PRINT$STRING(.D);
276   2            CALL DE$HASH(DESIG, .CHAR);
277   2            CHAR(2), CHAR(3) = '$';
278   2            CALL CRT$PRINT$STRING(.CHAR);
279   2            END DISPLAY$DESIG;


                /***************************************************************
              -  ******************
                *
                * DISPLAY$TYPE:
                *   THIS PROCEDURE IS USED TO DISPLAY THE TYPE OF A CONTACT.
                *
                * PARAMETERS:
                *   - A.- POINTER TO A BYTE VALUE REPRESENTING THE TYPE TO BE
              -    DISPLAYED.
                *
                ****************************************************************
              -  ******************/
280   1         DISPLAY$TYPE: PROCEDURE (A) PUBLIC;
281   2            DCL A ADDRESS,
                   TYPE BASED A BYTE;
 282   2        DCL S  (*) BYTE DATA ('SURFACE    $$'),
                   SS (*) BYTE DATA ('SUB-SURFACE$$'),
                   T (10) BYTE DATA ('TYPE:   $$');

 283   2        CALL CRT$PRINT$STRING(.T);
 284   2        IF TYPE = 0
                  THEN CALL CRT$PRINT$STRING(.S);
 286   2          ELSE CALL CRT$PRINT$STRING(.SS);
 287   2        END DISPLAY$TYPE;


              /**************************************************************
          -    *****************
              *
              * DISPLAY$CLASS:
              *   THIS PROCEDURE IS USED TO DISPLAY THE CLASS OF A CONTACT.
              *
              * PARAMETERS:
              *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE CLASS TO
          -    BE DISPLAYED IS
              *     LOCATED.
              *
              **************************************************************
          -    *****************/
 288   1      DISPLAY$CLASS: PROCEDURE (A) PUBLIC;
 289   2        DCL A ADDRESS,
                    CLASS BASED A BYTE;
 290   2        DCL FRI (*) BYTE DATA (' FRIEND$$'),
                         HOS (*) BYTE DATA ('HOSTILE$$'),
                         UNK (*) BYTE DATA ('UNKNOWN$$'),
                         C (10) BYTE DATA ('CLASS:  $$');

 291   2             CALL CRT$PRINT$STRING(.C);
 292   2             DO CASE CLASS;
 293   3                CALL CRT$PRINT$STRING(.FRI);
 294   3                CALL CRT$PRINT$STRING(.HOS);
 295   3                CALL CRT$PRINT$STRING(.UNK);
 296   3                END;
 297   2             END DISPLAY$CLASS;


                    /**********************************************************
               -     ******************
                    *
                    * DISPLAY$LAT$LONG:
                    *   THIS PROCEDURE IS USED TO DISPLAY THE VALUES OF LATITUDE
               -     AND LONGITUDE.
                    *
                    * PARAMETERS:
                    *   - KIND.- DENOTES LATITUDE IF 0, OTHERWISE DENOTES LONGITU
               -     DE.
                    *   - A.- POINTER TO THE FP REPRESENTATION OF LAT/LONG.
                    *
                    **********************************************************
               -     ******************/
 298   1            DISPLAY$LAT$LONG: PROCEDURE (KIND, A) PUBLIC;
 299   2               DCL A ADDRESS,
                       VALUE BASED A (4) BYTE,
                       CHAR (9) BYTE,
                       (KIND, I) BYTE;
 300   2           DCL LAT (14) BYTE DATA ('LATITUDE:   $$'),
                       LONG (14) BYTE DATA ('LONGITUDE:  $$'),
                       NORTH (8) BYTE DATA (' NORTH$$'),
                       SOUTH (8) BYTE DATA (' SOUTH$$'),
                       EAST  (8) BYTE DATA (' EAST $$'),
                       WEST  (8) BYTE DATA (' WEST $$');

 301   2           IF KIND = 0
                   THEN DO;
 303   3                CALL CRT$PRINT$STRING(.LAT);
 304   3                CALL LAT$LONG$FORMAT(A, .CHAR, 0);
 305   3                END;
 306   2           ELSE DO;
 307   3                CALL CRT$PRINT$STRING(.LONG);
 308   3                CALL LAT$LONG$FORMAT(A, .CHAR, 1);
 309   3                END;
 310   2           DO I = 0 TO 5;
 311   3              IF I = 3 THEN CALL CRT$WRITE(COLON);
 313   3              IF I = 5 THEN CALL CRT$WRITE(POINT);
 315   3              CALL CRT$WRITE(CHAR(I));
 316   3              END;
 317   2           IF KIND = 0
                   THEN DO;
 319   3                IF CHAR(6) = 'N'
                           THEN CALL CRT$PRINT$STRING(.NORTH);
 321   3                   ELSE CALL CRT$PRINT$STRING(.SOUTH);
 322   3                END;
 323   2         ELSE DO;
 324   3              IF CHAR(6) = 'E'
                         THEN CALL CRT$PRINT$STRING(.EAST);
                         ELSE CALL CRT$PRINT$STRING(.WEST);
 326   3              END;
 327   3
 328   2          END DISPLAY$LAT$LONG;


                 /*****************************************************************
               -  ****************
                 *
                 * DISPLAY$XY:
                 *    THIS PROCEDURE IS USED TO DISPLAY VALUES OF 'X,Y' COORDIN
               -  ATES.
                 *
                 * PARAMETERS:
                 *    - TYPE.- IF 0 'X' WILL BE DISPLAYED, OTHERWISE 'Y' WILL B
               -  E DISPLAYED.
                 *    - A.- POINTER TO A FP REPRESENTATION OF THE X/Y VALUE.
                 *
                 *****************************************************************
               -  ****************/
 329   1         DISPLAY$XY: PROCEDURE (TYPE, A) PUBLIC;
 330   2             DCL A ADDRESS,
                         CHAR (14) BYTE,
                         (TYPE, TEMP, I) BYTE;
 331   2             DCL X (9) BYTE DATA ('X:     $$'),
                         Y (9) BYTE DATA ('Y:     $$');
```

```
332   2          IF TYPE = 0
                    THEN CALL CRT$PRINT$STRING(.X);
334   2             ELSE CALL CRT$PRINT$STRING(.Y);
335   2          TEMP = FP$FORMAT(A, .CHAR, 10, 2);
336   2          IF TEMP = 0
                    THEN CALL CRT$WRITE(PLUS$SIGN);
338   2             ELSE CALL CRT$WRITE(MINUS$SIGN);
339   2          DO I = 0 TO 11;
340   3             IF I = 10 THEN CALL CRT$WRITE(POINT);
342   3             CALL CRT$WRITE(CHAR(I));
343   3             END;
344   2          END DISPLAY$XY;


               /*************************************************************
            -   *****************
               *
               * DISPLAY$CRS$BRG:
               *   THIS PROCEDURE IS USED TO DISPLAY THE VALUES OF COURSE AN
            -   D BEARING.
               *
               * PARAMETERS:
               *    - KIND.- IF 0 THE VALUE OF COURSE WILL BE DISPLAYED, OTHE
            -   RWISE THE VALUE
               *      OF BEARING WILL BE DISPLAYED.
               *    - A.- POINTER TO THE FP REPRESENTATION OF COURSE/BEARING.
               *
               **************************************************************
            -   *****************/
```

```
 345   1      DISPLAY$CRS$BRG: PROCEDURE (KIND, A) PUBLIC;
 346   2         DCL A ADDRESS,
                     CHAR (6) BYTE,
                     (I, TEMP, KIND) BYTE;
 347   2         DCL CRS (12) BYTE DATA ('COURSE:   $$'),
                     BRG (12) BYTE DATA ('BEARING:  $$');

 348   2         IF KIND = 0
                    THEN CALL CRT$PRINT$STRING(.CRS);
                    ELSE CALL CRT$PRINT$STRING(.BRG);
 350   2         TEMP = FP$FORMAT(A, .CHAR, 3, 1);
 351   2         DO I = 0 TO 3;
 352   2            IF I = 3 THEN CALL CRT$WRITE(POINT);
 353   3            CALL CRT$WRITE(CHAR(I));
 355   3            END;
 356   3         END DISPLAY$CRS$BRG;
 357   2


             /*****************************************************************
              ****************
             *
             * DISPLAY$SPD:
             *    THIS PROCEDURE IS USED TO DISPLAY THE VALUE OF SPEED.
             *
             * PARAMETERS:
             *    - A.- POINTER TO THE FP REPRESENTATION OF THE VALUE OF SP
              EED.
             *
             ******************************************************************
```

```
                  -    ****************/
 358    1         DISPLAY$SPD: PROCEDURE (A) PUBLIC;
 359    2            DCL A ADDRESS,
                         CHAR (5) BYTE,
                         (TEMP, I) BYTE;
 360    2            DCL SPD (12) BYTE DATA ('SPEED:    $$');

 361    2            CALL CRT$PRINT$STRING(.SPD);
 362    2            TEMP = FP$FORMAT(A, .CHAR, 2, 1);
 363    2            DO I = 0 TO 2;
 364    3               IF I = 2 THEN CALL CRT$WRITE(POINT);
 366    3               CALL CRT$WRITE(CHAR(I));
 367    3               END;
 368    2            END DISPLAY$SPD;


                  /****************************************************************
                  -  ****************
                  *
                  * DISPLAY$RANGE:
                  *    THIS PROCEDURE IS USD TO DISPLAY THE VALUE OF RANGE.
                  *
                  * PARAMETERS:
                  *    - A.- POINTER TO THE FP REPRESENTATION OF RANGE.
                  *
                  ****************************************************************
                  -  *****************/
 369    1         DISPLAY$RANGE: PROCEDURE (A) PUBLIC;
 370    2            DCL A ADDRESS,
                     CHAR (8) BYTE,
                     I BYTE;
 371    2         DCL MLS (8) BYTE DATA (' MILES$$'),
                     YDS (8) BYTE DATA (' YARDS$$'),
                     RNG (12) BYTE DATA ('RANGE:    $$');

 372    2         CALL CRT$PRINT$STRING(.RNG);
 373    2         CALL RANGE$FORMAT(A, .CHAR);
 374    2         DO I = 0 TO 4;
 375    3            CALL CRT$WRITE(CHAR(I));
 376    3            END;
 377    2         IF CHAR(5) = 'M'
                     THEN CALL CRT$PRINT$STRING(.MLS);
 379    2            ELSE CALL CRT$PRINT$STRING(.YDS);
 380    2         END DISPLAY$RANGE;


              /************************************************************
    -          ******************
              *
              * DISPLAY$TIME:
              *    THIS PROCEDURE IS USED TO DISPLAY THE VALUE OF THE TIME.
              *
              * PARAMETERS:
              *    - A.- POINTER TO A 3 BYTE VECTOR CONTAINING THE TIME VALU
    -          E.
              *
              ************************************************************
    -          **************/
```

```
381   1      DISPLAY$TIME: PROCEDURE (A) PUBLIC;
382   2        DCL A ADDRESS,
                   DIGIT BASED A BYTE,
                   CHAR BYTE;
383   2        DCL TIME (12) BYTE DATA ('TIME:      $$');

384   2        CALL CRT$PRINT$STRING(.TIME);
385   2        CHAR = DIGIT/10 + 30H;
386   2        CALL CRT$WRITE(CHAR);
387   2        CHAR = DIGIT MOD 10 + 30H;
388   2        CALL CRT$WRITE(CHAR);
389   2        A = A + 1;
390   2        CALL CRT$WRITE(COLON);
391   2        CHAR = DIGIT/10 + 30H;
392   2        CALL CRT$WRITE(CHAR);
393   2        CHAR = DIGIT MOD 10 + 30H;
394   2        CALL CRT$WRITE(CHAR);
395   2        A = A + 1;
396   2        CALL CRT$WRITE(COLON);
397   2        CHAR = DIGIT/10 + 30H;
398   2        CALL CRT$WRITE(CHAR);
399   2        CHAR = DIGIT MOD 10 + 30H;
400   2        CALL CRT$WRITE(CHAR);
401   2        END DISPLAY$TIME;


             /***********************************************************************
              *****************
             *
```

```
                  * DISPLAY$ORIGIN:
                  *   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE C
               -  OORDINATE GRID
                  *   ORIGIN.
                  *
                  ***********************************************************
               -   ****************/
 402   1          DISPLAY$ORIGIN: PROCEDURE PUBLIC;
 403   2             DCL MSG (*) BYTE DATA
                         ('THE COORDINATE GRID ORIGIN VALUES ARE:$$');

 404   2             CALL CRT$PRINT$STRING(.TITLE$0);
 405   2             CALL SEND$CRLF;
 406   2             CALL CRT$PRINT$STRING(.MSG);
 407   2             CALL SEND$CRLF;
 408   2             CALL DISPLAY$LAT$LONG(0, .SYSTEM.LAT);
 409   2             CALL SEND$CRLF;
 410   2             CALL DISPLAY$LAT$LONG(1, .SYSTEM.LONG);
 411   2             CALL SEND$CRLF;
 412   2             CALL SEND$CRLF;
 413   2             CALL CHECK$GO$KEY;
 414   2             CALL CLEAR$LOW$SCREEN;
 415   2             END DISPLAY$ORIGIN;


                  /**********************************************************
               -   ****************
                  *
                  * DISPLAY$SCALE:
```

```
                  *   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE S
                -  CALE BEING USED
                  *   IN THE GRAPHICS DISPLAY.
                  *
                  ****************************************************************
                -  *****************/
 416    1         DISPLAY$SCALE: PROCEDURE PUBLIC;
 417    2            DCL CHAR (6) BYTE,
                         (I, TEMP) BYTE;
 418    2            DCL MSG (*) BYTE DATA
                             ('THE VALUE OF THE GRAPHICS SCALE IS:  $$');

 419    2            CALL CRT$PRINT$STRING(.TITLE$1);
 420    2            CALL SEND$CRLF;
 421    2            CALL SEND$CRLF;
 422    2            CALL CRT$PRINT$STRING(.MSG);
 423    2            TEMP = FP$FORMAT(.SYSTEM.SCALE, .CHAR, 2, 2);
 424    2            DO I = 0 TO 3;
 425    3               IF I = 2 THEN CALL CRT$WRITE(POINT);
 427    3               CALL CRT$WRITE(CHAR(I));
 428    3               END;
 429    2            CALL SEND$CRLF;
 430    2            CALL SEND$CRLF;
 431    2            CALL CHECK$GO$KEY;
 432    2            CALL CLEAR$LOW$SCREEN;
 433    2            END DISPLAY$SCALE;


                  /***************************************************************
```

```
                 -   *****************
                     *
                     * DISPLAY$OWN$SHIP:
                     *   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE O
                 -   WN SHIP.
                     *
                     *************************************************************
                 -   ***************/
  434   1            DISPLAY$OWN$SHIP: PROCEDURE PUBLIC;
  435   2              DCL POINTER BYTE;
  436   2              CALL CRT$PRINT$STRING(.TITLE$2);
  437   2              CALL SEND$CRLF;
  438   2              CALL CRT$PRINT$STRING(.MSG$00);
  439   2              CALL SEND$CRLF;
  440   2              CALL DISPLAY$LAT$LONG(0, .OWN$SHIP$INFO.LAT);
  441   2              CALL SEND$SPACE(12);
  442   2              CALL DISPLAY$LAT$LONG(1, .OWN$SHIP$INFO.LONG);
  443   2              CALL SEND$CRLF;
  444   2              POINTER = OWN$SHIP$INFO.POINTER;
  445   2              CALL DISPLAY$XY(0, .OWN$SHIP(POINTER).X);
  446   2              CALL SEND$SPACE(18);
  447   2              CALL DISPLAY$XY(1, .OWN$SHIP(POINTER).Y);
  448   2              CALL SEND$CRLF;
  449   2              CALL SEND$CRLF;
  450   2              CALL CHECK$GO$KEY;
  451   2              CALL CLEAR$LOW$SCREEN;
  452   2              CALL CRT$PRINT$STRING(.TITLE$2);
  453   2              CALL SEND$CRLF;
  454   2              CALL CRT$PRINT$STRING(.MSG$01);
  455   2              CALL DISPLAY$TIME(.OWN$SHIP(POINTER).TIME);
```

```
456   2         CALL SEND$CRLF;
457   2         CALL DISPLAY$CRS$BRG(0, .OWN$SHIP(POINTER).CRS);
458   2         CALL SEND$SPACE(20);
459   2         CALL DISPLAY$SPD(.OWN$SHIP(POINTER).SPD);
460   2         CALL SEND$CRLF;
461   2         CALL SEND$CRLF;
462   2         CALL CHECK$GO$KEY;
463   2         CALL CLEAR$LOW$SCREEN;
464   2         END DISPLAY$OWN$SHIP;


               /****************************************************************
          -     ****************
               *
               * DISPLAY$CONTACT$INFO:
               *   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT ANY C
          -     ONTACT BEING
               *   HANDLED BY THE SYSTEM.
               *
               *****************************************************************
          -     ****************/
465   1        DISPLAY$CONTACT$INFO: PROCEDURE PUBLIC;
466   2           DCL DESIG ADDRESS,
                      LAT (4) BYTE,
                      LONG (4) BYTE,
                      TIME$1 (4) BYTE , TIME$2 (4) BYTE ,
                      X$DELTA (4) BYTE, Y$DELTA (4) BYTE,
                      COS$DIST (4) BYTE, SIN$DIST (4) BYTE,
                      BRG (4) BYTE, RNG (4) BYTE,
                   T (3) BYTE, DISTANCE (4) BYTE,
                   STRING (23) BYTE,
                   (TEMP, OK, INDEX, I, POINTER) BYTE;
 467   2         DCL DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH); /*
         -     0.0174532925 */

 468   2         DCL MSG0 (*) BYTE DATA
                         ('NO COURSE INFORMATION AVAILABLE.$$'),
                     MSG1 (*) BYTE DATA
                         ('NO SPEED INFORMATION AVAILABLE.$$'),
                     MSG2 (*) BYTE DATA
                         ('NO CPA INFORMATION AVAILABLE.$$'),
                     MSG3 (*) BYTE DATA
                         ('ENTER CONTACT DESIG AS REQUESTED:$$'),
                     MSG4 (*) BYTE DATA
                         ('DESIG NOT IN USE.$$'),
                     MSG5 (*) BYTE DATA
                         ('CURRENT NUMBER OF POSITIONS:  $$'),
                     MSG6 (*) BYTE DATA
                         ('THE ACTUAL ESTIMATED POSITION IS:$$');

 469   2         T(0) = HOURS;    /* SAVE TIME OF CALL */
 470   2         T(1) = MINUTES;
 471   2         T(2) = SECONDS;
 472   2         OK = 0;
 473   2         DO WHILE OK = 0;
 474   3            CALL CRT$PRINT$STRING(.TITLE$3);
 475   3            CALL SEND$CRLF;
 476   3            CALL CRT$PRINT$STRING(.MSG3);
 477   3            CALL SEND$CRLF;
```

```
3            DESIG = GET$DESIG;
3            INDEX = CHECK$DESIG(DESIG);
3            IF INDEX = ØFFH
             THEN DO;
4                 CALL CRT$PRINT$STRING(.MSG4);
4                 CALL SEND$CRLF;
4                 CALL CHECK$GO$KEY;
4                 END;
3            ELSE DO;
4                 OK = 1;
4                 END;
3            CALL CLEAR$LOW$SCREEN;
3            END;
2         CALL CRT$PRINT$STRING(.TITLE$3);
2         CALL SEND$CRLF;
2         CALL CRT$PRINT$STRING(.MSG$Ø3);
2         CALL SEND$CRLF;
2         CALL DISPLAY$DESIG(DESIG);
2         CALL SEND$SPACE(15);
2         CALL DISPLAY$TYPE(.CONTACT$INFO(INDEX).TYPE);
2         CALL SEND$SPACE(6);
2         CALL DISPLAY$CLASS(.CONTACT$INFO(INDEX).KIND);
2         CALL SEND$CRLF;
2         CALL CRT$PRINT$STRING(.MSG5);
2         IF CONTACT$INFO(INDEX).FLAG
             THEN TEMP = 15;
2            ELSE TEMP = (CONTACT$INFO(INDEX).POINTER  MOD 15) + 1;
2         CALL BYTE$CHAR(TEMP);
2         CALL SEND$CRLF;
2         CALL SEND$CRLF;
```

```
508    2          CALL CHECK$GO$KEY;
509    2          CALL CLEAR$LOW$SCREEN;
510    2          CALL CRT$PRINT$STRING(.TITLE$3);
511    2          CALL SEND$CRLF;
512    2          CALL CRT$PRINT$STRING(.MSG$00);
513    2          CALL SEND$CRLF;
514    2          POINTER = CONTACT$INFO(INDEX).POINTER;
515    2          CALL CONV$LAT$LONG(.CONTACT$POSI(POINTER).X, .CONTACT$POSI
              -  (POINTER).Y,
                                       .LAT, .LONG);
516    2          CALL DISPLAY$LAT$LONG(0, .LAT);
517    2          CALL SEND$SPACE(12);
518    2          CALL DISPLAY$LAT$LONG(1, .LONG);
519    2          CALL SEND$CRLF;
520    2          CALL DISPLAY$XY(0, .CONTACT$POSI(POINTER).X);
521    2          CALL SEND$SPACE(18);
522    2          CALL DISPLAY$XY(1, .CONTACT$POSI(POINTER).Y);
523    2          CALL SEND$CRLF;
524    2          CALL SEND$CRLF;
525    2          CALL CHECK$GO$KEY;
526    2          CALL CLEAR$LOW$SCREEN;
527    2          TEMP = GET$CPA(INDEX, .STRING);
528    2          CALL CRT$PRINT$STRING(.TITLE$3);
529    2          CALL SEND$CRLF;
530    2          CALL CRT$PRINT$STRING(.MSG$01);
531    2          CALL DISPLAY$TIME(.CONTACT$POSI(POINTER).TIME);
532    2          CALL SEND$CRLF;
533    2          CALL DISPLAY$CRS$BRG(1, .CONTACT$POSI(POINTER).BRG);
534    2          CALL SEND$SPACE(20);
535    2          CALL DISPLAY$RANGE(.CONTACT$POSI(POINTER).RNG);
```

```
 536    2          CALL SEND$CRLF;
 537    2          IF CONTACT$INFO(INDEX).CRS$FLAG
                   THEN DO;
 539    3               CALL DISPLAY$CRS$BRG(0, .CONTACT$POSI(POINTER).CRS);
 540    3               CALL SEND$SPACE(20);
 541    3               END;
 542    2          ELSE DO;
 543    3               CALL CRT$PRINT$STRING(.MSG0);
 544    3               CALL SEND$SPACE(3);
 545    3               END;
 546    2          IF CONTACT$INFO(INDEX).SPD$FLAG
                      THEN CALL DISPLAY$SPD(.CONTACT$POSI(POINTER).SPD);
 548    2             ELSE CALL CRT$PRINT$STRING(.MSG1);
 549    2          CALL SEND$CRLF;
 550    2          CALL SEND$CRLF;
 551    2          CALL CHECK$GO$KEY;
 552    2          CALL CLEAR$LOW$SCREEN;
 553    2          CALL CRT$PRINT$STRING(.TITLE$3);
 554    2          CALL SEND$CRLF;
 555    2          CALL CRT$PRINT$STRING(.MSG$02);
 556    2          CALL SEND$CRLF;
 557    2          IF TEMP = 0
                   THEN DO;
 559    3               CALL CRT$PRINT$STRING(.MSG2);
 560    3               CALL SEND$CRLF;
 561    3               END;
 562    2          ELSE DO;
 563    3               IF STRING(14) < 3AH
                        THEN DO;
 565    4                    CALL CRT$PRINT$STRING(.('TIME:     $$'));
```

```
566   4              DO I = 7 TO 10;
567   5                IF I = 9 THEN CALL CRT$WRITE(COLON);
569   5                CALL CRT$WRITE(STRING(I));
570   5                END;
571   4              CALL SEND$SPACE(9);
572   4              CALL CRT$PRINT$STRING(.('BEARING:  $$'));
573   4              DO I = 11 TO 14;
574   5                IF I = 14 THEN CALL CRT$WRITE(POINT);
576   5                CALL CRT$WRITE(STRING(I));
577   5                END;
578   4              CALL SEND$SPACE(9);
579   4              CALL CRT$PRINT$STRING(.('RANGE:    $$'));
580   4              DO I = 15 TO 19;
581   5                CALL CRT$WRITE(STRING(I));
582   5                END;
583   4              IF STRING(20) = 'M'
                       THEN CALL CRT$PRINT$STRING(.(' MILES$$'));
585   4                ELSE CALL CRT$PRINT$STRING(.(' YARDS$$'));
586   4              CALL SEND$CRLF;
587   4              END;
588   3           ELSE DO;
589   4              CALL SEND$SPACE(30);
590   4              IF STRING(14) = 'L'
                     THEN DO;
592   5                   CALL SEND$SPACE(5);
593   5                   CALL START$BLINK;
594   5                   CALL CRT$PRINT$STRING(.('COLLISION AT $$'))
            -   ;
595   5                   STRING(12), STRING(13) = '$';
596   5                   STRING(11) = STRING(10);
```

```
597    5                        STRING(10) = STRING(9);
598    5                        STRING(9) = COLON;
599    5                        CALL CRT$PRINT$STRING(.STRING(7));
600    5                        CALL CRT$WRITE(18H);         /* STOP BLINK
          -    */
601    5                        END;
602    4                    ELSE DO;
603    5                        IF STRING(14) = 'A'
                                  THEN CALL CRT$PRINT$STRING(.('    MOVING
          -     AWAY.$$'));
605    5                          ELSE CALL CRT$PRINT$STRING(.('SAME COURS
          -    E AND SPEED.$$'));
606    5                        END;
607    4                    END;
608    3                END;
609    2            CALL SEND$CRLF;
610    2            CALL CHECK$GO$KEY;
611    2            CALL CLEAR$LOW$SCREEN;
612    2            CALL CRT$PRINT$STRING(.TITLE$3);
613    2            CALL SEND$CRLF;
614    2            CALL CRT$PRINT$STRING(.MSG6);
615    2            CALL SEND$CRLF;
616    2            IF CONTACT$INFO(INDEX).CRS$FLAG AND
                       CONTACT$INFO(INDEX).SPD$FLAG
                    THEN DO;
618    3                IF T(0) < CONTACT$POSI(POINTER).TIME(0)
                           THEN T(0) = T(0) + 24;
                           /* FIND ESTIMATED TIME */
620    3                CALL CONV$CONTACT$TIME(.T, .TIME$1);
621    3                CALL CONV$CONTACT$TIME(.CONTACT$POSI(POINTER).TIME, .
```

AD-A102 693 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
MICROCOMPUTER BASED SHIP-BOARD GUN CONTROL SYSTEM.(U)
MAR 81 A ERDOGAN

UNCLASSIFIED NL

5 OF 7
AD
A102693

```
               -   TIME$2);
 622    3                CALL FSUB(.TIME$1, .TIME$2, .TIME$1);
                           /* FIND ESTIMATED DISTANCE TRAVELED BY CONTACT */
 623    3                CALL FMUL(.CONTACT$POSI(POINTER).SPD, .TIME$1, .DISTA
               -   NCE);
 624    3                CALL FMUL(.DEG$TO$RAD, .CONTACT$POSI(POINTER).CRS, .B
               -   RG);
 625    3                CALL COS$SIN(.BRG, .COS$DIST, .SIN$DIST);
 626    3                CALL FMUL(.DISTANCE, .COS$DIST, .Y$DELTA);
 627    3                CALL FMUL(.DISTANCE, .SIN$DIST, .X$DELTA);
 628    3                CALL FADD(.Y$DELTA, .CONTACT$POSI(POINTER).Y, .Y$DELT
               -   A);
 629    3                CALL FADD(.X$DELTA, .CONTACT$POSI(POINTER).X, .X$DELT
               -   A);
 630    3                TEMP = OWN$SHIP$INFO.POINTER;
 631    3                CALL FSUB(.Y$DELTA, .OWN$SHIP(TEMP).Y, .Y$DELTA);
 632    3                CALL FSUB(.X$DELTA, .OWN$SHIP(TEMP).X, .X$DELTA);
                           /* FIND ESTIMATED BEARING */
 633    3                CALL ARC$TAN(.Y$DELTA, .X$DELTA, .BRG);
 634    3                CALL FDIV(.BRG, .DEG$TO$RAD, .BRG);
                           /* FIND ESTIMATED RANGE */
 635    3                CALL FSQR(.Y$DELTA, .Y$DELTA);
 636    3                CALL FSQR(.X$DELTA, .X$DELTA);
 637    3                CALL FADD(.Y$DELTA, .X$DELTA, .RNG);
 638    3                CALL FSQRT(.RNG, .RNG);
                           /* PRINT ESTIMATED VALUES */
 639    3                CALL DISPLAY$CRS$BRG(1, .BRG);
 640    3                CALL SEND$SPACE(20);
 641    3                CALL DISPLAY$RANGE(.RNG);
 642    3                END;
```

```
643   2          ELSE DO;
644   3               CALL CRT$PRINT$STRING(.MSG0);
645   3               CALL SEND$SPACE(3);
646   3               CALL CRT$PRINT$STRING(.MSG1);
647   3               END;
648   2          CALL SEND$CRLF;
649   2          CALL SEND$CRLF;
650   2          CALL CHECK$GO$KEY;
651   2          CALL CLEAR$LOW$SCREEN;
652   2          END DISPLAY$CONTACT$INFO;


               /**************************************************************
            -   ***************
               *
               * DISPLAY$SYSTEM:
               *   THIS PROCEDURE IS USED TO DISPLAY THE DESIGNATIONS OF ALL
            -    THE CONTACTS
               *   THAT ARE BEING MAINTAINED BY THE SYSTEM.
               *
               **************************************************************
            -   ****************/
653   1        DISPLAY$SYSTEM: PROCEDURE PUBLIC;
654   2           DCL BUFFER (18) BYTE,
                      I BYTE;
655   2           DCL M0 (*) BYTE DATA
                         ('THE FOLLOWING CONTACTS ARE BEING MAINTAINED BY THE
            -     SYSTEM:$$');
```

```
 656    2          DO I = 0 TO LAST(BUFFER);
 657    3             BUFFER(I) = ' ';
 658    3             END;
 659    2          BUFFER(16), BUFFER(17) = '$';
 660    2          CALL CRT$PRINT$STRING(.TITLE$4);
 661    2          CALL SEND$CRLF;
 662    2          CALL CRT$PRINT$STRING(.M0);
 663    2          CALL SEND$CRLF;
 664    2          DO I = 0 TO 14;
 665    3             IF CONTACT$INFO(I).DESIG <> 00H
                      THEN DO;
 667    4                   CALL DB$HASH(CONTACT$INFO(I).DESIG, .BUFFER(10));
 668    4                   CALL CRT$PRINT$STRING(.BUFFER);
 669    4                   END;
 670    3              END;
 671    2          CALL SEND$CRLF;
 672    2          CALL CHECK$GO$KEY;
 673    2          CALL CLEAR$LOW$SCREEN;
 674    2          END DISPLAY$SYSTEM;


         /************************************************************
      -   **************
         *
         * DISPLAY$SAFE$RNG:
         *   THIS PROCEDURE IS USED TO PRESENT TO THE OPERATOR THE CUR
      -   RENT VALUE OF
         *   THE SAFE C.P.A. RANGE.
         *
```

```
                  *****************************************************************
              -    ****************/
 675    1        DISPLAY$SAFE$RNG: PROCEDURE PUBLIC;
 676    2           CALL CRT$PRINT$STRING(.TITLE$5);
 677    2           CALL SEND$CRLF;
 678    2           CALL SEND$CRLF;
 679    2           CALL DISPLAY$RANGE(.SAFE$RNG);
 680    2           CALL SEND$CRLF;
 681    2           CALL SEND$CRLF;
 682    2           CALL CHECK$GO$KEY;
 683    2           CALL CLEAR$LOW$SCREEN;
 684    2           END DISPLAY$SAFE$RNG;


                 /*****************************************************************
              -    ***************
                 *
                 * DISPLAY$WIND:
                 *   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE W
              -    IND.
                 *
                 ******************************************************************
              -    ***************/
 685    1        DISPLAY$WIND: PROCEDURE PUBLIC;
 686    2           DCL STRING (6) BYTE,
                        (TEMP, I) BYTE;
 687    2           DCL MSG0 (*) BYTE DATA ('WIND DIRECTION:  $$'),
                        MSG1 (*) BYTE DATA ('WIND SPEED:       $$');

 688    2           CALL CRT$PRINT$STRING(.TITLE$6);
```

```
CALL SEND$CRLF;
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG0);
TEMP = FP$FORMAT(.SYSTEM.WIND$DIR, .STRING, 3, 1);
DO I = 0 TO 3;
   IF I = 3 THEN CALL CRT$WRITE(POINT);
   CALL CRT$WRITE(STRING(I));
   END;
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG1);
TEMP = FP$FORMAT(.SYSTEM.WIND$SPD, .STRING, 2, 1);
DO I = 0 TO 2;
   IF I = 2 THEN CALL CRT$WRITE(POINT);
   CALL CRT$WRITE(STRING(I));
   END;
CALL SEND$CRLF;
CALL SEND$CRLF;
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
END DISPLAY$WIND;


/*****************************************************************
 ***************
 *
 * DISPLAY$UPDATE$TIME:
 *   THIS PROCEDURE IS USED TO DISPLAY THE TIME BETWEEN UPDATE
 S.
 *
```

```
                    ******************************************************************
                 -   ****************/
 711   1            DISPLAY$UPDATE$TIME: PROCEDURE (TEMP$TIME) PUBLIC;
 712   2               DCL TEMP$TIME BYTE;
 713   2               DCL M0 (*) BYTE DATA
                              ('THE TIME BETWEEN UPDATES IS $$'),
                           M1 (*) BYTE DATA
                              (' SECONDS.$$');

 714   2               CALL CRT$PRINT$STRING(.TITLE$7);
 715   2               CALL SEND$CRLF;
 716   2               CALL CRT$PRINT$STRING(.M0);
 717   2               CALL BYTE$CHAR(TEMP$TIME);
 718   2               CALL CRT$PRINT$STRING(.M1);
 719   2               CALL SEND$CRLF;
 720   2               CALL SEND$CRLF;
 721   2               CALL CHECK$GO$KEY;
 722   2               CALL CLEAR$LOW$SCREEN;
 723   2               END DISPLAY$UPDATE$TIME;


 724   1            END DISPLAY$CMDS;


MODULE INFORMATION:

     CODE AREA SIZE     = 11B0H   4528D
     VARIABLE AREA SIZE = 00D2H    210D
```

```
PL/M-80 COMPILER                                        20 MAR 81   PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE CPAMODULE
OBJECT MODULE PLACED IN CPA.OBJ
COMPILER INVOKED BY:  :F1:PLM80 CPA.SRC PAGELENGTH(33) PAGEWIDTH(75) DATE(2
                    -0 MAR 81)


   1            CPA$MODULE: DO;
                $NOLIST


 252   1        DCL SAFE$RNG (4) BYTE EXTERNAL;




                /****************************************************************
           -     **************
                *
                * CONV$CONTACT$TIME:
                *   THIS PROCEDURE IS USED TO CONVERT A GIVEN CONTACT TIME (I
           -     N HOURS,
                *   MINUTES, AND SECONDS) INTO A FP REPRESENTATION (IN HOURS
           -     AND TENTHS
                *   OF HOURS).
                *
                * PARAMETERS:
                *   - S.- POINTER TO A MEMORY LOCATION IN WHICH THE TIME IS I
           -     OCATED
```

```
                  *         (IN HOURS, MINUTES, AND SECONDS).
                  *   - T.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
                -  REPRESEN-
                  *         TING THE TIME IS DESIRED TO BE PLACED.
                  *
                  ************************************************************
                -  ***************/
 253   1          CONV$CONTACT$TIME: PROCEDURE (S, T) PUBLIC;
 254   2            DCL (S, T) ADDRESS,
                        STRING BASED S (3) BYTE,
                        TIME$FLOAT BASED T (4) BYTE,
                        TEMP (4) BYTE,
                        FP$60 (4) BYTE DATA (00H,00H,70H,42H),
                        FP$3600 (4) BYTE DATA (00H,00H,61H,45F);
 255   2            TEMP(0) = STRING(0);
 256   2            TEMP(1), TEMP(2), TEMP(3) = 00H;
 257   2            CALL FLTDS (.TEMP, .TIME$FLOAT);
 258   2            TEMP(0) = STRING(1);
 259   2            TEMP(1), TEMP(2), TEMP(3) = 00H;
 260   2            CALL FLTDS (.TEMP, .TEMP);
 261   2            CALL FDIV (.TEMP, .FP$60, .TEMP);
 262   2            CALL FADD (.TEMP, .TIME$FLOAT, .TIME$FLOAT);
 263   2            TEMP(0) = STRING(2);
 264   2            TEMP(1), TEMP(2), TEMP(3) = 00H;
 255   2            CALL FLTDS (.TEMP, .TEMP);
 266   2            CALL FDIV (.TEMP, .FP$3600, .TEMP);
 257   2            CALL FADD (.TEMP, .TIME$FLOAT, .TIME$FLOAT);
 268   2            END CONV$CONTACT$TIME;
```

```
                /**************************************************************
        -        ****************
                 *
                 * CPA$TIME$CONV:
                 *    THIS PROCEDURE IS CALLED BY THE "CPA$CALCULATION" PROCEDU
        -        RE IN
                 *    ORDER TO CONVERT A FP REPRESENTATION OF THE CPA TIME TO A
        -         STRING OF
                 *    ASCII CHARACTERS.
                 *
                 * PARAMETERS:
                 *    - T.- POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRES
        -        ENTATION OF
                 *          THE CPA TIME IS LOCATED.
                 *    - S.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF
        -         CHARACTERS RE-
                 *          PRESENTING THE CPA TIME IS DESIRED TO BE PLACED.
                 *
                 **************************************************************
        -        *****************/
 269   1        CPA$TIME$CONV: PROCEDURE (T, S);
 270   2          DCL (T, S) ADDRESS,
                      FP$60 (4) BYTE DATA (00H,00H,70H,42H),  /* 60.0 */
                      CPA$TIME BASED T (4) BYTE,
                      STRING BASED S (4) BYTE,
                      (HOURS, MINUTES, TEMP) (4) BYTE,
                      J BYTE;
 271   2          CALL FIXSD (.CPA$TIME, .HOURS);
 272   2          DO J = 0 TO 3;
 273   3            TEMP(J) = HOURS(J);
 274   3            END;
 275   2         DO WHILE HOURS(0) >= 24;
 276   3            HOURS(0) = HOURS(0) - 24;
 277   3            END;
 278   2         STRING(0) = HOURS(0) / 10 + 30H;
 279   2         STRING(1) = HOURS(0) MOD 10 + 30H;
 280   2         CALL FLTDS (.TEMP, .TEMP);
 281   2         CALL FSUB (.CPASTIME, .TEMP, .MINUTES);
 282   2         CALL FMUL (.MINUTES, .FP$60, .MINUTES);
 283   2         CALL FIXSD (.MINUTES, .MINUTES);
 284   2         IF MINUTES(0) >= 60
                   THEN DO;
 286   3              MINUTES(0) = MINUTES(0) - 60;
 287   3              HOURS(0) = HOURS(0) + 1;
 288   3              IF HOURS(0) >= 24 THEN HOURS(0) = HOURS(0) - 24;
 290   3              STRING(0) = HOURS(0) / 10 + 30H;
 291   3              STRING(1) = HOURS(0) MOD 10 + 30H;
 292   3              END;
 293   2         STRING(2) = MINUTES(0) / 10 + 30H;
 294   2         STRING(3) = MINUTES(0) MOD 10 + 30H;
 295   2         END CPASTIME$CONV;


                   /****************************************************************
                    ***************
                   *
                   * CONTACT$CRS$SPD:
```

```
              *   THIS PROCEDURE IS USED TO CALCULATE THE COURSE AND SPEED
            -  OF A CONTACT
              *   GIVEN ITS LAST KNOWN POSITION AT "CONTACT$POSI" STRUCTURE
            -  .
              *   THIS PROCEDURE IS CALLED BY THE "CPA$CALCULATION" PROCEDU
            -  RE.
              *
              * PARAMETERS:
              *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE RELATIVE
            -  COURSE OF A
              *         CONTACT IS LOCATED (IN RADIANS).
              *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE RELATIVE
            -  SPEED OF  A
              *         CONTACT IS LOCATED (IN KNOTS).
              *   - S.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF
            -   CHARACTERS
              *         REPRESENTING THE TRUE COURSE AND SPEED OF A CONTACT
            -   IS DESIRED
              *         TO BE PLACED.
              *   - INDEX.- IT IS THE VALUE WHICH GIVES THE LAST KNOWN POSI
            -  TION  IN
              *             THE "CONTACT$POSI" STRUCTURE OF A CONTACT BEING
            -   PROCESSED
              *             BY THE "CPA$CALCULATION" PROCEDURE.
              *
              *******************************************************************
            -  ****************/
 296   1      CONTACT$CRS$SPD: PROCEDURE (A, B, S, INDEX);
 297   2         DCL (A, B, S) ADDRESS,
                      CRS BASED A (4) BYTE,
                       SPD BASED B (4) BYTE,
                       STRING BASED S (9) BYTE,
                       FP$DEG$TO$RAD (4) BYTE DATA (35H,0FAH,0EH,3CH),  /* 0.
                 -   0174532925  */
                       (TEMP, TEMP1) (4) BYTE,
                       (SIN$CRS, COS$CRS) (4) BYTE,
                       (X1, X2, XM, Y1, Y2, YM) (4) BYTE,
                       (INDEX, I, J) BYTE;
 298    2            J = OWN$SHIP$INFO.POINTER;
 299    2            DO I = 0 TO 3;
 300    3              TEMP(I) = OWN$SHIP(J).CRS(I);
 301    3              TEMP1(I) = OWN$SHIP(J).SPD(I);
 302    3              END;
 303    2            CALL FMUL (.TEMP, .FP$DEG$TO$RAD, .TEMP);
 304    2            CALL COS$SIN (.TEMP, .COS$CRS, .SIN$CRS);
 305    2            CALL FMUL (.TEMP1, .SIN$CRS, .X1);
 306    2            CALL FMUL (.TEMP1, .COS$CRS, .Y1);
 307    2            CALL COS$SIN (.CRS, .COS$CRS, .SIN$CRS);
 308    2            CALL FMUL (.SPD, .SIN$CRS, .X2);
 309    2            CALL FMUL (.SPD, .COS$CRS, .Y2);
 310    2            CALL FADD (.X1, .X2, .XM);
 311    2            CALL FADD (.Y1, .Y2, .YM);
 312    2            CALL FSQR (.XM, .TEMP);
 313    2            CALL FSQR (.YM, .TEMP1);
 314    2            CALL FADD (.TEMP, .TEMP1, .TEMP);
 315    2            CALL FSQRT (.TEMP, .TEMP);    /*  TRUE SPEED   */
 316    2            CALL ARC$TAN (.YM, .XM, .TEMP1);
 317    2            CALL FDIV (.TEMP1, .FP$DEG$TO$RAD, .TEMP1);   /*  TRUE COU
                 -   RSE IN DEGREES  */
 318    2            DO I = 0 TO 3;
 319    3              CONTACT$POSI(INDEX).CRS(I) = TEMP1(I);
 320    3              CONTACT$POSI(INDEX).SPD(I) = TEMP(I);
 321    3              END;
 322    2           J = INDEX/15;
 323    2           CONTACT$INFO(J).CRS$FLAG , CONTACT$INFO(J).SPD$FLAG = 2FFH
             -     ;
 324    2           J = FP$FORMAT (.TEMP1, .STRING(0), 3, 1);
 325    2           J = FP$FORMAT (.TEMP, .STRING(4), 2, 1);
 326    2           END CONTACT$CRS$SPD;


                  /*****************************************************************
             -     ***************
                  *
                  * CPA$CALCULATION:
                  *    THIS PROCEDURE IS CALLED BY THE "GET$CPA" PROCEDURE IN OR
             -     DER TO
                  *    CALCULATE THE CPA OF A GIVEN CONTACT.
                  *    THE "LEAST SQUARE FIT" METHOD IS USED WITH AT MOST 5 POSI
             -     TIONS OF
                  *    A GIVEN CONTACT.
                  *
                  * PARAMETERS:
                  *    - INDEX1.- IT INDICATES THE FIRST POSITION AT THE "CONTAC
             -     T$POSI"
                  *       STRUCTURE THAT HAS THE INFORMATION ABOUT THE GIVEN CONT
             -     ACT.
                  *    - INDEX2.- IT INDICATES THE FIRST POSITION AT THE "CONTAC
             -     T$POSI"
```

```
                 *     STRUCTURE THAT WILL BE USED IN THE "LEAST SQUARE FIT" C
               -  OMPUTATION.
                 *   - COUNT.- IT GIVES THE COUNTING USED TO DETERMINE THE NUM
               -  BER OF CONTACT
                 *     POSITIONS TO BE USED IN THE CALCULATION OF THE CPA.
                 *   - S.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF
               -   CHARACTERS
                 *     REPRESENTING THE CPA INFORMATION IS DESIRED TO BE PLACE
               -  D.
                 *
                 *************************************************************
               -  *****************/
 327   1         CPA$CALCULATION: PROCEDURE (INDEX1, INDEX2, COUNT, S);
 328   2           DCL S ADDRESS,
                       STRING BASED S (23) BYTE,
                       CHECK BYTE DATA (04H),             /* CHECK FOR EQUAL *
               -  /
                       CHECK1 BYTE DATA (02H),            /* CHECK FOR GREATER
               -   THAN */
                       CHECK2 BYTE DATA (05H),            /* CHECK FOR NOT EQU
               -  AL */
                       FP$DEG$TO$RAD (4) BYTE DATA (35H,0FAH,8EH,3CH), /* 0.0
               -  174532925 */
                       FP$1 (4) BYTE DATA (00H,00H,80H,3FH),           /* 1.0
               -   */
                       FP$60 (4) BYTE DATA (00H,00H,70H,42H),          /* 60.
               -  0 */
                       PI$OVER2 (4) BYTE DATA (0DBH,0FH,0C9H,3FH),     /* 1.5
               -  707963 */
                       PI$FLOAT (4) BYTE DATA (0DBH,0FH,49H,40H),      /* 3.1
```

```
-   41593 */
           PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H),    /* 4.7
-   123889 */
           MSG0 (*) BYTE DATA
                 ('SAME CPS & SPD'),
           MSG1 (*) BYTE DATA
                 (' COLLISION'),
           MSG2 (*) BYTE DATA
                 ('MOVING AWAY    '),
           REL$XY (5) STRUCTURE
                    ( X(4) BYTE,
                      Y(4) BYTE),
           COS$BRG (4) BYTE, SIN$BRG (4) BYTE,
           X1 (4) BYTE, Y1 (4) BYTE,
           SLOPE (4) BYTE, Y$CUT (4) BYTE,
           BIG$Y1 (4) BYTE, BIG$Y2 (4) BYTE,
           X$CPA (4) BYTE, Y$CPA (4) BYTE,
           CPA$TIME (4) BYTE, CPA$BRG (4) BYTE, CPA$RNG (4) BYTE,
           TIME (4) BYTE, TIME1 (4) BYTE,
           S0 (4) BYTE, S1 (4) BYTE, S2 (4) BYTE,
           T0 (4) BYTE, T1 (4) BYTE,
           X$SQUARE (4) BYTE, XY$PROD (4) BYTE,
           S1$SQUARE (4) BYTE, ST$PROD (4) BYTE,
           NUMERATOR (4) BYTE, DENOMINATOR (4) BYTE,
           (REL$CRS, REL$SPD, TEMP$RAD) (4) BYTE,
           (INDEX1,INDEX2,COUNT,TEMP,TEMP1,I,J,FLAG,FLAG1,LAST$PO
-   INTEP) BYTE;

           /* LAST$POINTER WILL POINT TO THE LAST POSITION IN CONT
-   ACT$POSI  */
```

```
 329    2          TEMP = INDEX1 / 15;
 330    2          LAST$POINTER = CONTACT$INFO(TEMP).POINTER;
 331    2          TEMP = INDEX2;
 332    2          DO I = 0 TO 3;
 333    3             S0(I), S1(I), S2(I), T0(I), T1(I) = 00H;
 334    3             END;
                              /* COMPUTE PARAMETERS FOR LEAST SQUARE FIT */
 335    2          S0(0) = COUNT + 1;
 336    2          CALL FLTDS (.S0, .S0);
 337    2          DO I = 0 TO COUNT;
 338    3             IF TEMP > INDEX1 + 14 THEN TEMP = INDEX1;
 340    3             CALL FMUL (.CONTACT$POSI(TEMP).BRG, .FP$DEGS$TO$RAD, .TE
            -       MP$RAD);
 341    3             CALL COS$SIN (.TEMP$RAD, .COS$BRG,.SIN$BRG);
 342    3             CALL FMUL (.CONTACT$POSI(TEMP).RNG,.SIN$BRG,.REL$XY(I).
            -       X);
 343    3             CALL FMUL (.CONTACT$POSI(TEMP).RNG,.COS$BRG,.REL$XY(I).
            -       Y);
 344    3             CALL FADD (.REL$XY(I).X, .S1, .S1);
 345    3             CALL FSQR (.REL$XY(I).X, .X$SQUARE);
 346    3             CALL FADD (.X$SQUARE, .S2, .S2);
 347    3             CALL FADD (.REL$XY(I).Y, .T0, .T0);
 348    3             CALL FMUL (.REL$XY(I).Y, .REL$XY(I).X, .XY$PROD);
 349    3             CALL FADD (.XY$PROD, .T1, .T1);
 350    3             TEMP = TEMP + 1;
 351    3             END;
 352    2          CALL FMUL (.S0, .S2, .DENOMINATOR);
 353    2          CALL FSQR (.S1, .S1$SQUARE);
 354    2          CALL FSUB (.DENOMINATOR, .S1$SQUARE, .DENOMINATOR);
 355    2          CALL FMUL (.S0, .T1, .NUMERATOR);
```

```
356   2          CALL FMUL (.S1, .T0, .ST$PROD);
357   2          CALL FSUB (.NUMERATOR, .ST$PROD, .NUMERATOR);
358   2          FLAG = FZTST (.DENOMINATOR, .CHECK);
359   2          FLAG1 = FZTST (.NUMERATOR, .CHECK);
360   2          IF FLAG AND FLAG1 THEN I = 0;                  /* SLOPE = 0/0 *
          -    /
362   2          IF FLAG AND (NOT FLAG1) THEN I = 1;            /* SLOPE = X/0 *
          -    /
364   2          IF (NOT FLAG) AND FLAG1 THEN I = 2;            /* SLOPE = 0 */
365   2          IF (NOT FLAG) AND (NOT FLAG1) THEN I = 3;  /* SLOPE <> 0 */
368   2          TEMP = LASTSPOINTER;
369   2          IF TEMP = INDEX1
                    THEN TEMP1 = TEMP + 14;
371   2             ELSE TEMP1 = LAST$POINTER - 1;
372   2          IF (I = 0) AND (FLAG1 := FCMPR(.CONTACT$POSI(TEMP).RNG,
                                                    .CONTACT$POSI(TEMP1).RNG, .
          -    CHECK2))
                    THEN I = 1;
374   2          IF (I = 1) OR (I = 3)
                 THEN DO;
376   3                IF FCMPR(.CONTACT$POSI(TEMP).BRG,.CONTACT$POSI(TEMP1)
          -    .BRG,.CHECK) AND
                          FCMPR(.CONTACT$POSI(TEMP).RNG,.CONTACT$POSI(TEMP1)
          -    .RNG,.CHECK)
                       THEN I = 0;
378   3                END;
379   2          DO CASE I;
380   3             DO;          /* CONTACT ON SAME CRS & SPD OF OWN SHIP *
          -    /
381   4                TEMP = LAST$POINTER;
```

```
 382    4              TEMP1 = OWN$SHIP$INFO.POINTER;
 383    4              DO J = 0 TO 3;
 384    5                 CONTACT$POSI(TEMP).CRS(J) = OWN$SHIP(TEMP1).CRS(J
           -    );
 385    5                 CONTACT$POSI(TEMP).SPD(J) = OWN$SHIP(TEMP1).SPD(J
           -    );
 386    5                 END;
 387    4              TEMP1 = FP$FORMAT(.CONTACT$POSI(TEMP).CRS, .STRING(
           -    ), 3, 1);
 388    4              TEMP1 = FP$FORMAT(.CONTACT$POSI(TEMP).SPD, .STRING(4
           -    ), 2, 1);
 389    4              TEMP1 = INDEX1 / 15;
 390    4              CONTACT$INFO(TEMP1).CRS$FLAG = 0FFH;
 391    4              CONTACT$INFO(TEMP1).SPD$FLAG = 0FFH;
 392    4              DO J = 7 TO LAST(STRING);
 393    5                 STRING(J) = MSG0(J - 7);
 394    5                 END;
 395    4           END;
 396    3           DO;           /* CONTACT RELATIVE COURSE = 000 OR 180 */
 397    4              TEMP = LAST$POINTER;
 398    4              IF TEMP = INDEX1
                          THEN TEMP1 = TEMP + 14;
 400    4                 ELSE TEMP1 = LAST$POINTER - 1;
 401    4              FLAG = FCMPR(.CONTACT$POSI(TEMP1).RNG,.CONTACT$POSI(
           -    TEMP).RNG,.CHECK1);
 402    4              CALL FMUL (.CONTACT$POSI(INDEX2).BRG, .FP$DEG$TO$RAD
           -    , .TEMP$RAD);
 403    4              CALL COS$SIN(.TEMP$RAD, .COS$BRG,.SIN$BRG);
 404    4              CALL FMUL(.CONTACT$POSI(INDEX2).RNG,.SIN$BRG,.X$CPA)
           -    ;
 405   4               IF X$CPA(3) >= 80H
                          THEN X$CPA(3) = X$CPA(3) XOR 080H;
                             /* CONVERT TIME TO FP */
 407   4               TEMP = LAST$POINTER;
 408   4               CALL CONV$CONTACT$TIME(.CONTACT$POSI(INDEX2).TIME, .
            -   TIME);
 409   4               IF CONTACT$POSI(INDEX2).TIME(0) >
                          CONTACT$POSI(TEMP).TIME(0)
                       THEN DO;
 411   5                  CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).T
            -   IME(0) + 24;
 412   5                  CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME,.
            -   TIME1);
 413   5                  CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).T
            -   IME(0) - 24;
 414   5                  END;
 415   4               ELSE CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME
            -   ,.TIME1);
                             /* COMPUTE RELATIVE COURSE */
 416   4               IF (FLAG1 := FCMPR(.REL$XY(COUNT).Y, .REL$XY(0).Y, .
            -   CHECK1))
                       THEN DO;
 418   5                    DO J = 0 TO 3;
 419   5                       REL$CRS(J) = 00H;
 420   5                       END;
 421   5                    END;
 422   4               ELSE DO;
 423   5                    DO J = 0 TO 3;
 424   5                       REL$CRS(J) = PI$FLOAT(J);
 425   5                       END;
```

```
426   5                  END;
                            /* COMPUTE RELATIVE SPEED */
427   4               CALL FSUB (.TIME1, .TIME, .TIME1);
428   4               CALL FSUB (.REL$XY(COUNT).Y,.REL$XY(0).Y, .Y1);
429   4               IF Y1(3) >= 80H THEN Y1(3) = Y1(3) XOR 80H;
431   4               CALL FDIV (.Y1, .TIME1, .REL$SPD);
                            /* COMPUTE TRUE COURSE AND SPEED */
432   4               TEMP = LAST$POINTER;
433   4               CALL CONTACT$CRS$SPD (.REL$CRS, .REL$SPD, .STRING, T
            -   EMP);
434   4               IF FLAG
                      THEN DO;         /* CONTACT CLOSING */
                               /* COMPUTE CPA TIME */
436   5                  CALL FMUL (.CONTACT$POSI(INDEX2).RNG,.COS$BRG,.Y$
            -   CPA);
437   5                  CALL FDIV (.Y$CPA, .REL$SPD, .CPA$TIME);
438   5                  IF CPA$TIME(3) >= 080H
                            THEN CPA$TIME(3) = CPA$TIME(3) XOR 080H;
440   5                  CALL FADD(.CPA$TIME, .TIME, .CPA$TIME);
441   5                  CALL CPA$TIME$CONV (.CPA$TIME, .STRING(7));
                               /* CHECK FOR COLLISION */
442   5                  IF (FLAG1 := FCMPR (.SAFE$RNG, .X$CPA, .CHECK1))
                         THEN DO;
444   6                     DO J = 11 TO LAST(STRING);
445   7                        STRING(J) = MSG1(J - 11);
446   7                        END;
447   5                     RETURN;
448   6                     END;
                               /* COMPUTE CPA BEARING */
449   5                  CALL FMUL (.CONTACT$POSI(INDEX2).BRG, .FP$DEG$TOS
                -   RAD, .TEMP$RAD);
                             IF (FLAG1 := FCMPR(.PI$FLOAT,.TEMP$RAD, .CHECK1))
 450   5                     THEN DO;
                                STRING(11) = '0';
 452   5                        STRING(12) = '9';
 453   6                        STRING(13) = '0';
 454   6                        STRING(14) = '0';
 455   5                        END;
 456   6                     ELSE DO;
 457   5                        STRING(11) = '2';
 458   6                        STRING(12) = '7';
 459   5                        STRING(13) = '0';
 460   6                        STRING(14) = '0';
 461   5                        END;
 462   6                           /* COMPUTE CPA RANGE */
                             CALL RANGE$FORMAT (.X$CPA, .STRING(15));
 463   5                     END;
 464   5                  ELSE DO;  /* CONTACT MOVING AWAY */
 465   4                     DO J = 7 TO LAST(STRING);
 466   5                        STRING(J) = MSG2(J - 7);
 467   6                        END;
 468   6                     END;
 469   5           END;
 470   4           DO;            /* CONTACT RELATIVE COURSE = 090 OR 270 *
 471   3
                -   /
                      TEMP = LASTSPOINTER;
 472   4              IF TEMP = INDEX1
 473   4                 THEN TEMP1 = TEMP + 14;
                         ELSE TEMP1 = LAST$POINTER - 1;
 475   4              FLAG = FCMPR(.CONTACT$POSI(TEMP1).RNG, .CONTACT$POSI
 476   4
```

```
                 -   (TEMP).RNG,
                                           .CHECK1);
 477    4                  CALL FMUL (.CONTACT$POSI(INDEX2).BRG, .FP$DEG$TO$RAD
                 -   , .TEMP$RAD);
 478    4                  CALL COS$SIN(.TEMP$RAD, .COS$BRG,.SIN$BRG);
 479    4                  CALL FMUL(.CONTACT$POSI(INDEX2).RNG,.COS$BRG,.Y$CPA
                 -   ;
 480    4                  IF Y$CPA(3) >= 080H
                              THEN Y$CPA(3) = Y$CPA(3) XOR 080H;
                                              /* CONVERT TIME TO FP */
 482    4                  TEMP = LAST$POINTER;
 483    4                  CALL CONV$CONTACT$TIME(.CONTACT$POSI(INDEX2).TIME,.
                 -   IME);
 484    4                  IF CONTACT$POSI(INDEX2).TIME(0) >
                              CONTACT$POSI(TEMP).TIME(0)
                           THEN DO;
 486    5                     CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).T
                 -   IME(0) + 24;
 487    5                     CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME,.
                 -   TIME1);
 488    5                     CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).T
                 -   IME(0) - 24;
 489    5                     END;
 490    4                  ELSE CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME
                 -   ,.TIME1);
                                  /* COMPUTE RELATIVE COURSE */
 491    4                  IF(FLAG1:= FCMPR (.REL$XY(COUNT).X, .REL$XY(0).X, .C
                 -   HECK1))
                           THEN DO;
 493    5                     DO J = 0 TO 3;
 494   5                        REL$CRS(J) = PI$OVER2(J);
 495   6                        END;
 496   5                     END;
 497   4                ELSE DO;
 498   5                     DO J = 0 TO 3;
 499   6                        REL$CRS(J) = PI$3$OVER2(J);
 500   5                        END;
 501   5                     END;
                             /* COMPUTE RELATIVE SPEED */
 502   4                CALL FSUB (.TIME1, .TIME, .TIME1);
 503   4                CALL FSUB (.REL$XY(COUNT).X,.REL$XY(0).X, .X1);
 504   4                IF X1(3) >= 80H THEN X1(3) = X1(3) XOR 80H;
 506   4                CALL FDIV(.X1, .TIME1, .REL$SPD);
                                /* COMPUTE TRUE COURSE AND SPEED */
 507   4                TEMP = LAST$POINTER;
 508   4                CALL CONTACT$CRS$SPD (.REL$CRS, .REL$SPD, .STRING, T
            -    EMP);
 509   4                IF FLAG
                        THEN DO;        /* CONTACT CLOSING */
                                    /* COMPUTE CPA TIME */
 511   5                   CALL FMUL(.CONTACT$POSI(INDEX2).RNG, .SIN$BRG, .X
            -    $CPA);
 512   5                   CALL FDIV(.X$CPA, .REL$SPD, .CPA$TIME);
 513   5                   IF CPA$TIME(3) >= 080H
                              THEN CPA$TIME(3) = CPA$TIME(3) XOR 080H;
 515   5                   CALL FADD(.CPA$TIME, .TIME, .CPA$TIME);
 516   5                   CALL CPA$TIME$CONV (.CPA$TIME, .STRING(7));
                                    /* CHECK FOR COLLISION */
 517   5                   IF (FLAG1 := FCMPR (.SAFE$RNG, .Y$CPA, .CHECK1))
                           THEN DO;
 519   6                      DO J = 11 TO LAST(STRING);
 520   7                         STRING(J) = MSG1(J - 11);
 521   7                         END;
 522   5                      RETURN;
 523   6                      END;
                                 /* COMPUTE CPA BEARING */
 524   5                CALL FMUL(.CONTACT$POSI(INDEX2).BRG, .FP$DEG$TOSR
          -    AD, .TEMP$RAD);
 525   5                IF (FLAG := FCMPR(.PI$3$OVER2,.TEMP$RAD, .CHECK1)
          -    )
                        AND (FLAG:= FCMPR(.TEMP$RAD, .PI$OVER2,.CHECK1))
                        THEN DO;
 527   6                   STRING(11) = '1';
 528   6                   STRING(12) = '8';
 529   5                   STRING(13) = '0';
 530   5                   STRING(14) = '0';
 531   5                   END;
 532   5                ELSE DO;
 533   6                   STRING(11) = '0';
 534   6                   STRING(12) = '0';
 535   6                   STRING(13) = '0';
 536   5                   STRING(14) = '0';
 537   6                   END;
                              /* COMPUTE CPA RANGE */
 538   5                CALL RANGE$FORMAT (.Y$CPA, .STRING(15));
 539   5                END;
 540   4             ELSE DO; /* CONTACT MOVING AWAY */
 541   5                DO J = 7 TO LAST(STRING);
 542   6                   STRING(J) = MSG2(J - 7);
 543   5                   END;
```

```
544   5                    END;
545   4                 END;
546   3                 DO;          /* RELATIVE MOVE - GENERAL CASE */
547   4                    TEMP = LAST$POINTER;
548   4                    IF TEMP = INDEX1
                              THEN TEMP1 = TEMP + 14;
550   4                       ELSE TEMP1 = LAST$POINTER - 1;
551   4                    FLAG = FCMPR(.CONTACT$POSI(TEMP1).RNG,.CONTACT$POSI(
            -   TEMP).RNG,
                                                .CHECK1);
                        /* PERFORM LEAST SQUARE FIT FOR LAST POSITIONS */
                                                   /* COMPUTE SLOPE */
552   4                    CALL FDIV (.NUMERATOR, .DENOMINATOR, .SLOPE);
                                                    /* COMPUTE CUT AT Y AXIS */
553   4                    CALL FMUL (.S2, .T0, .NUMERATOR);
554   4                    CALL FMUL (.S1, .T1, .ST$PROD);
555   4                    CALL FSUB (.NUMERATOR, .ST$PROD, .NUMERATOR);
556   4                    CALL FDIV (.NUMERATOR, .DENOMINATOR, .Y$CUT);
                             /* COMPUTE RELATIVE COURSE */
                             /* PREPARE TO COMPUTE RELATIVE SPEED */
557   4                    CALL FMUL(.SLOPE, .REL$XY(0).X, .BIG$Y1);
558   4                    CALL FADD(.BIG$Y1, .Y$CUT, .BIG$Y1);
559   4                    CALL FMUL(.SLOPE, .REL$XY(COUNT).X, .BIG$Y2);
560   4                    CALL FADD(.BIG$Y2, .Y$CUT, .BIG$Y2);
561   4                    CALL FSUB(.BIG$Y2, .BIG$Y1, .NUMERATOR);
562   4                    CALL FSUB(.REL$XY(COUNT).X, .REL$XY(0).X, .DENOMINAT
            -   OR);
563   4                    CALL ARC$TAN(.NUMERATOR, .DENOMINATOR, .REL$CRS);
564   4                    CALL FSQR(.NUMERATOR, .NUMERATOR);
565   4                    CALL FSQR(.DENOMINATOR, .DENOMINATOR);
```

```
566    4                CALL FADD(.NUMERATOR, .DENOMINATOR, .NUMERATOR);
567    4                CALL FSQRT(.NUMERATOR, .NUMERATOR);
                                          /* CONVERT TIME TO FP */
568    4                TEMP = LAST$POINTER;
569    4                CALL CONV$CONTACT$TIME(.CONTACT$POSI(INDEX2).TIME, .
           -   TIME);
570    4                IF CONTACT$POSI(INDEX2).TIME(0) >
                        CONTACT$POSI(TEMP).TIME(0)
                        THEN DO;
572    5                   CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).T
           -   IME(0) + 24;
573    5                   CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME,
           -   .TIME1);
574    5                   CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).T
           -   IME(0) - 24;
575    5                   END;
576    4                ELSE DO;
577    5                     CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME
           -   , .TIME1);
578    5                     END;
579    4                CALL FSUB(.TIME1, .TIME, .TIME1);
                                         /* COMPUTE RELATIVE SPEED */
580    4                CALL FDIV(.NUMERATOR, .TIME1, .REL$SPD);
                                  /* COMPUTE TRUE COURSE AND SPEED */
581    4                TEMP = LAST$POINTER;
582    4                CALL CONTACT$CRS$SPD (.REL$CRS, .REL$SPD, .STRING, T
           -   EMP);
583    4                IF FLAG
                        THEN DO;         /* CONTACT CLOSING */
                                             /* COMPUTE CPA */
```

```
 585   5          CALL FMUL(.SLOPE, .REL$XY(0).X, .NUMERATOR);
 586   5          CALL FSUB(.NUMERATOR, .BIG$Y1, .NUMERATOR);
 587   5          CALL FMUL(.SLOPE, .NUMERATOR, .NUMERATOR);
 588   5          CALL FSQR(.SLOPE, .DENOMINATOR);
 589   5          CALL FADD(.DENOMINATOR, .FP$1, .DENOMINATOR);
                                  /* COMPUTE X$CPA */
 590   5          CALL FDIV(.NUMERATOR, .DENOMINATOR, .X$CPA);
                                     /* COMPUTE Y$CPA */
 591   5          CALL FMUL(.SLOPE, .REL$XY(0).X, .NUMERATOR);
 592   5          CALL FSUB(.BIG$Y1, .NUMERATOR, .NUMERATOR);
 593   5          CALL FDIV(.NUMERATOR, .DENOMINATOR, .Y$CPA);
                                 /* COMPUTE CPA TIME */
 594   5          CALL FSUB(.X$CPA, .REL$XY(0).X, .NUMERATOR);
 595   5          CALL FSQR(.NUMERATOR, .NUMERATOR);
 596   5          CALL FSUB(.Y$CPA, .BIG$Y1, .DENOMINATOR);
 597   5          CALL FSQR(.DENOMINATOR, .DENOMINATOR);
 598   5          CALL FADD(.NUMERATOR, .DENOMINATOR, .NUMERATOR );
 599   5          CALL FSQRT(.NUMERATOR, .NUMERATOR);
 600   5          CALL FDIV(.NUMERATOR, .REL$SPD, .CPA$TIME);
 601   5          CALL FADD(.TIME, .CPA$TIME, .CPA$TIME);
 602   5          CALL CPA$TIME$CONV (.CPA$TIME, .STRING(7));
                                    /* COMPUTE CPA RANGE */
 603   5          CALL FSQR(.X$CPA, .NUMERATOR);
 604   5          CALL FSQR(.Y$CPA, .DENOMINATOR);
 605   5          CALL FADD(.NUMERATOR, .DENOMINATOR, .NUMERATOR);
 606   5          CALL FSQRT(.NUMERATOR, .CPA$RNG);
                                    /* CHECK FOR COLLISION */
 607   5          IF (FLAG1 := FCMPR (.SAFE$RNG, .CPA$RNG, .CHECK1)
         -   )
                    THEN DO;
```

```
609   6                         DO J = 11 TO LAST(STRING);
610   7                            STRING(J) = MSG1(J - 11);
611   7                            END;
612   5                         RETURN;
613   6                         END;
                                 /* COMPUTE CPA BEARING */
614   5                   CALL ARC$TAN(.Y$CPA, .X$CPA, .CPA$BRG);
615   5                   CALL CONV$RAD$MIN(.CPA$BRG, .CPA$BRG);
616   5                   CALL FDIV(.CPA$BRG, .FP$60, .CPA$BRG);
617   5                   TEMP = FP$FORMAT (.CPA$BRG, .STRING(11), 3, 1);
618   5                   CALL RANGE$FORMAT(.CPA$RNG, .STRING(15));
619   5                   END;
620   4                ELSE DO;
                                       /* CONTACT MOVING AWAY */
621   5                   DO J = 7 TO LAST(STRING);
622   6                      STRING(J) = MSG2(J - 7);
623   6                      END;
624   5                   END;
625   4             END;

626   3            END;    /* END CASE */
627   2          END CPA$CALCULATION;


           /**************************************************************
            **************
           *
           * GET$CPA:
           *   THIS PROCEDURE IS USED TO FIND THE CPA INFORMATION ABOUT
           -   A CONTACT.
               *
               * PARAMETERS:
               *    - INDEX.- INDICATES THE RELATIVE POSITION OF THE CONTACT
           -   IN THE
               *       CONTACT$INFO STRUCTURE.
               *    - A.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF
           -    CHARACTERS
               *       REPRESENTING THE CPA INFORMATION IS DESIRED TO BE PLACE
           -   D.
               *
               * USAGE:
               *    TYPED PROCEDURE. RETURNS A VALUE OF 0 IF NO CPA CAN BE DE
           -   TERMINED AT
               *    THE MOMENT. OTHERWISE, A VALUE OF 1 IS RETURNED.
               *
               ************************************************************
           -    ****************/
628   1        GET$CPA: PROCEDURE (INDEX, A) BYTE PUBLIC;
629   2           DCL A ADDRESS,
                      STRING BASED A (23) BYTE,
                      (INDEX, INDEX1, COUNT, TEMP, P1, P2, I) BYTE;
630   2           DO I = 0 TO LAST(STRING);
631   3              STRING(I) = 020H;         /*  PUT BLANKS IN WORK BUFFER
           -    */
632   3              END;
633   2           INDEX1 = 15*INDEX;
634   2           IF (CONTACT$INFO(INDEX).POINTER < INDEX1 + 1) AND
                     (NOT CONTACT$INFO(INDEX).FLAG)
                  THEN RETURN 0;
```

```
636   2        P1 = CONTACT$INFO(INDEX).POINTER;
637   2        P2 = CONTACT$INFO(INDEX).OS$POINTER;
638   2        IF CONTACT$INFO(INDEX).FLAG
               THEN DO;
640   3             IF P2 = 0FFH
                    THEN DO;
642   4                  COUNT = 4;
643   4                  IF P1 < INDEX1 + 4
                            THEN TEMP = P1 + 11;
645   4                     ELSE TEMP = P1 - 4;
646   4                  END;
647   3             ELSE DO;
648   4                  COUNT = 0;
649   4                  TEMP = P2 + 1;
650   4                  IF TEMP = (INDEX1 + 15)
                            THEN TEMP = INDEX1;
652   4                  DO WHILE TEMP <> P1;
653   5                     IF TEMP = (INDEX1 + 15)
                               THEN TEMP = INDEX1;
655   5                        ELSE TEMP = TEMP + 1;
656   5                     COUNT = COUNT + 1;
657   5                     END;
658   4                  IF COUNT = 0
                            THEN RETURN 0;
660   4                  TEMP = P2 + 1;
661   4                  IF TEMP = (INDEX1 + 15)
                            THEN TEMP = INDEX1;
663   4                  END;
664   3             END;
665   2        ELSE DO;
```

```
666   3           IF P2 = 0FFH
                  THEN DO;
668   4                COUNT = P1 - INDEX1;
669   4                IF COUNT > 4 THEN COUNT = 4;
671   4                TEMP = P1 - COUNT;
672   4                END;
673   3           ELSE DO;
674   4                IF P1 = P2 THEN RETURN 0;
676   4                COUNT = P1 - P2 - 1;
677   4                TEMP = P2 + 1;
678   4                IF COUNT = 0 THEN RETURN 0;
680   4                END;
681   3           END;
682   2      CALL CPA$CALCULATION(INDEX1, TEMP, COUNT, .STRING);
683   2      RETURN 1;
684   2      END GET$CPA;



685   1    END CPA$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 152DH   5421D
     VARIABLE AREA SIZE = 00FBH    251D
     MAXIMUM STACK SIZE = 000AH     10D
     1205 LINES READ
```

```
                                                  20 MAR 81  PAGE   1
PL/M-80 COMPILER

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PLASMAMODULE
OBJECT MODULE PLACED IN PLASMA.OBJ
COMPILER INVOKED BY:  :F1:PLM80 PLASMA.SRC PAGELENGTH(33) PAGEWIDTH(75) DAT
                      -E(20 MAR 81)



   1            PLASMA$MODULE: DO;


                $NOLIST

                                      /***   DECLARATIONS:   ***/


 286   1        DCL TRUE LIT '0FFH',
                    FALSE LIT '00H',
                    EOL LIT '024H',
                    POINT LIT '02EH';

 287   1        DCL FP$2   (4) BYTE DATA (000H, 000H, 000H, 040H),   /* 2.0 *
            -   /
                    FP$8   (4) BYTE DATA (0CDH, 0CCH, 008H, 041H),   /* 8.55
            -   */
                    FP$511 (4) BYTE DATA (000H, 000H, 0FFH, 043H);   /* 511.0
            -   */
```

```
PL/M-80 COMPILER                                        20 MAR 81  PAGE   2

288   1      DCL LAST$POSI (15) STRUCTURE(
                                FLAG BYTE,
                                X ADDRESS,
                                Y ADDRESS),

                OS$LAST$POSI STRUCTURE(
                             FLAG BYTE,
                             X ADDRESS,
                             Y ADDRESS);

289   1      DCL WINDOW (4) BYTE,
                HALF$WINDOW (4) BYTE,
                ORIGIN$X (4) BYTE,
                ORIGIN$Y (4) BYTE;

290   1      DCL BUFFER (*) BYTE INITIAL ('SCALE:          ');

             /*************************************************************
       -      ***************
             *
             * SET$WINDOW:
             *   THIS PROCEDURE IS USED TO FIND THE VALUES OF THE WINDOW A
       -      ND HALF$WINDOW
```

```
                *   FLOATING POINT VECTORS.
                *
                ****************************************************************
             -   ***************/
 291   1        SET$WINDOW: PROCEDURE PUBLIC;
 292   2           CALL FMUL(.SYSTEM.SCALE, .FP$8, .WINDOW);
 293   2           CALL FDIV(.WINDOW, .FP$2, .HALF$WINDOW);
 294   2           END SET$WINDOW;


                /***************************************************************
             -   ****************
                *
                * CLEAR$STRUCTURES:
                *   THIS PROCEDURE IS USED TO CLEAR THE STRUCTURES USED IN TH
             -   IS MODULE.
                *
                ****************************************************************
             -   ***************/
 295   1        CLEAR$STRUCTURES: PROCEDURE PUBLIC;
 296   2           DCL A ADDRESS,
                       VALUE BASED A BYTE,
                       I BYTE;
 297   2           A = .LAST$POSI;
 298   2           DO I = 0 TO 74;
 299   3              VALUE = 00H;
 300   3              A = A + 1;
 301   3              END;
 302   2           A = .OS$LAST$POSI;
```

```
303   2        DO I = 0 TO 4;
304   3          VALUE = 00H;
305   3          A = A + 1;
306   3          END;
307   2        END CLEAR$STRUCTURES;


             /*************************************************************
          -   ******************
             *
             * DRAW$FRIENDLY$SYMBOL:
             *   THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY. IN
          -   A GIVEN POSITION,
             *   THE SIMBOL USED TO REPRESENT A FRIENDLY CONTACT.
             *
             * PARAMETERS:
             *   - X.- ADDRESS VALUE.
             *   - Y.- ADDRESS VALUE.
             *
             **************************************************************
          -   ******************/
308   1      DRAW$FRIENDLY$SYMBOL: PROCEDURE (X, Y) PUBLIC;
309   2        DCL (X, Y) ADDRESS,
                   (TEMP$X, TEMP$Y) ADDRESS;
310   2        IF (X < 2) OR (X > 509) OR (Y < 2) OR (Y > 509)
                  THEN RETURN;    /* SYMBOL CAN NOT BE DRAWN */
312   2        TEMP$X = X - 1;
313   2        TEMP$Y = Y + 2;
314   2        CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
```

```
315   2          TEMP$X = X + 1;
316   2          CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
317   2          TEMP$X = X + 2;
318   2          TEMP$Y = Y + 1;
319   2          CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
320   2          TEMP$Y = Y - 1;
321   2          CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
322   2          TEMP$X = X + 1;
323   2          TEMP$Y = Y - 2;
324   2          CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
325   2          TEMP$X = X - 1;
326   2          CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
327   2          TEMP$X = X - 2;
328   2          TEMP$Y = Y - 1;
329   2          CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
330   2          TEMP$Y = Y + 1;
331   2          CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
332   2          END DRAW$FRIENDLY$SYMBOL;


                /************************************************************
           -     ****************
                 *
                 * DRAW$UNK$HOS$SYMBOL:
                 *   THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY, IN
           -     A GIVEN POINT,
                 *   THE SYMBOL USED TO REPRESENT UNKNOWN AND/OR HOSTILE CONTA
           -     CTS.
                 *
```

```
                  * PARAMETERS:
                  *   - X.- ADDRESS VALUE.
                  *   - Y.- ADDRESS VALUE.
                  *
                  ****************************************************************
               -   ***************/
 333    1         DRAW$UNK$HOS$SYMBOL: PROCEDURE (X, Y) PUBLIC;
 334    2            DCL (X, Y, TEMP$X, TEMP$Y) ADDRESS;
 335    2            IF (X < 2) OR (X > 509) OR (Y < 2) OR (Y > 509)
                        THEN RETURN;    /* SYMBOL CAN NOT BE DRAWN. */
 337    2            TEMP$X = X - 2;
 338    2            TEMP$Y = Y - 2;
 339    2            CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
 340    2            TEMP$X = X + 2;
 341    2            TEMP$Y = Y + 2;
 342    2            CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
 343    2            TEMP$X = X - 2;
 344    2            CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
 345    2            TEMP$X = X + 2;
 346    2            TEMP$Y = Y - 2;
 347    2            CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
 348    2            END DRAW$UNK$HOS$SYMBOL;


                  /***************************************************************
               -   ****************
                  *
                  * DRAW$OWN$SHIP$SYMBOL:
                  *   THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY, IN
               -   A GIVEN POINT,
                *    THE SYMBOL USED TO REPRESENT THE OWN SHIP.
                *
                * PARAMETERS:
                *    - X.- ADDRESS VALUE.
                *    - Y.- ADDRESS VALUE.
                *
                ************************************************************
               -  **************/
 349   1        DRAW$OWN$SHIP$SYMBOL: PROCEDURE (X, Y) PUBLIC;
 350   2           DCL (X, Y, TEMP$X, TEMP$Y) ADDRESS;
 351   2           CALL DRAW$FRIENDLY$SYMBOL(X, Y);
 352   2           IF (X = 0) OR (X = 511) OR (Y = 0) OR (Y = 511)
                      THEN RETURN;    /* SYMBOL CAN NOT BE DRAWN. */
 354   2           TEMP$X = X - 1;
 355   2           TEMP$Y = Y + 1;
 356   2           CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
 357   2           TEMP$X = X + 1;
 358   2           CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
 359   2           CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
 360   2           TEMP$Y = Y - 1;
 361   2           CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
 362   2           CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
 363   2           TEMP$X = X - 1;
 364   2           CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
 365   2           CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
 366   2           TEMP$Y = Y + 1;
 367   2           CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
 368   2           END DRAW$OWN$SHIP$SYMBOL;
```

```
                 /*****************************************************************
              -   ****************
                 *
                 * CHECK$PLASMA:
                 *   THIS PROCEDURE IS USED TO DETERMINE WHETHER OR NOT A POIN
              -   T REPRESENTED
                 *   BY TWO X ,Y FP VALUES, CAN BE DISPLAYED AT THE PLASMA.
                 *
                 * PARAMETERS:
                 *   - A.- POINTER TO A FOUR BYTE VECTOR REPRESENTING THE X VA
              -   LUE;
                 *   - B.- POINTER TO A FOUR BYTE VECTOR REPRESENTING THE Y VA
              -   LUE.
                 *
                 * USAGE:
                 *   TYPED PROCEDURE. RETURNS A VALUE OF 'TRUE' (0FFH) IF THE
              -   POINT CAN BE
                 *   DISPLAYED AT THE CURRENT SCALE; OTHERWISE, A VALUE OF 'FA
              -   LSE' (00H)
                 *   IS RETURNED.
                 *
                 *****************************************************************
              -   ****************/
 369   1         CHECK$PLASMA: PROCEDURE (A, B) BYTE PUBLIC;
 370   2            DCL (A, B) ADDRESS,
                         X BASED A (4) BYTE,
                         Y BASED B (4) BYTE,
                         TEMP (4) BYTE,
                         CHECK BYTE DATA (02H);
 371   2              CALL FADD(.ORIGIN$X, .WINDOW, .TEMP);
 372   2              IF FCMPR(.X, .TEMP, .CHECK)
                         THEN RETURN FALSE;
 374   2              IF FCMPR(.ORIGIN$X, .X, .CHECK)
                         THEN RETURN FALSE;
 376   2              CALL FSUB(.ORIGIN$Y, .WINDOW, .TEMP);
 377   2              IF FCMPR(.Y, .ORIGIN$Y, .CHECK)
                         THEN RETURN FALSE;
 379   2              IF FCMPR(.TEMP, .Y, .CHECK)
                         THEN RETURN FALSE;
                         /*  EVERYTHING IS OKAY. RETURN A TRUE VALUE   */
 381   2              RETURN TRUE;
 382   2              END CHECK$PLASMA;


                  /***********************************************************
             -     ***************
                   *
                   * NORMALIZE:
                   *   THIS PROCEDURE IS USED TO NORMALIZE TWO FP VALUES INTO AD
             -     DRESS VALUES
                   *   THAT CAN BE USED TO DETERMINE A POINT IN THE PLASMA DISPL
             -     AY. IT MUST
                   *   BE USED AFTER 'CHECK$PLASMA'.
                   *
                   * PARAMETERS:
                   *   - A.- POINTER TO A FOUR BYTE VECTOR REPRESENTING A FP 'X'
             -      VALUE.
```

```
                *   - B.- POINTER TO A FOUR BYTE VECTOR REPRESENTING A FP 'Y'
              -    VALUE.
                *   - C.- POINTER TO AN ADDRESS VALUE IN WHICH THE 'NORMALIZE
              -   D X' IS
                *     DESIRED TO BE PLACED.
                *   - D.- POINTER TO AN ADDRESS VALUE IN WHICH THE 'NORMALIZE
              -   D Y' IS
                *     DESIRED TO BE PLACED.
                *
                ****************************************************************
              -  **************/
 383   1        NORMALIZE: PROCEDURE (A, B, C, D) PUBLIC;
 384   2          DCL (A, B, C, D) ADDRESS,
                      FP$X BASED A (4) BYTE,
                      FP$Y BASED B (4) BYTE,
                      X BASED C ADDRESS,
                      Y BASED D ADDRESS,
                      DELTA$X (4) BYTE,
                      DELTA$Y (4) BYTE,
                      TEMP (4) BYTE,
                      TEMP1 (4) BYTE;
 385   2          CALL FSUB(.FP$X, .ORIGIN$X, .DELTA$X);
 386   2          IF DELTA$X(3) >= 80H
                      THEN DELTA$X(3) = DELTA$X(3) XOR 080H;    /* ABSOLUTE VA
              -  LUE REQUIRED */
 388   2          CALL FSUB(.ORIGIN$Y, .FP$Y, .DELTA$Y);
 389   2          IF DELTA$Y(3) >= 80H
                      THEN DELTA$Y(3) = DELTA$Y(3) XOR 080H;    /* ABSOLUTE VA
              -  LUE REQUIRED */
 391   2          CALL FDIV(.FP$511, .WINDOW, .TEMP);
```

```
392   2         CALL FMUL(.TEMP, .DELTA$X, .TEMP1);
393   2         CALL FIXSD(.TEMP1, .TEMP1);
394   2         X = TEMP1(1);
395   2         X = SHL(X, 8) + TEMP1(0);
396   2         CALL FMUL(.TEMP, .DELTA$Y, .TEMP1);
397   2         CALL FIXSD(.TEMP1, .TEMP1);
398   2         Y = TEMP1(1);
399   2         Y = SHL(Y, 8) + TEMP1(0);
400   2         END NORMALIZE;


              /**************************************************************
           -   *****************
              *
              * PUT$OS$CENTER:
              *   THIS PROCEDURE IS USED TO CHANGE THE X/Y VALUES USED IN T
           -   HE PLASMA ORI-
              *   GIN, IN ORDER TO ALLOW THE OWN SHIP'S LAST POSITION TO BE
           -    IN THE NEW
              *   PLASMA DISPLAY CENTER.
              *
              ***************************************************************
           -   ****************/
401   1     PUT$OS$CENTER: PROCEDURE PUBLIC;
402   2         DCL POINTER BYTE;
403   2         POINTER = OWN$SHIP$INFO.POINTER;
404   2         CALL FSUB(.OWN$SHIP(POINTER).X, .HALF$WINDOW, .ORIGIN$X);
405   2         CALL FADD(.OWN$SHIP(POINTER).Y, .HALF$WINDOW, .ORIGIN$Y);
406   2         END PUT$OS$CENTER;
```

```
                   /************************************************************
                -   ****************
                   *
                   * PUT$CONTACT$CENTER:
                   *   THIS PROCEDURE IS USED TO CHANGE THE X/Y VALUES USED IN T
                -   HE PLASMA ORI-
                   *   GIN, IN ORDER TO ALLOW THE GIVEN CONTACT'LAST POSITION TO
                -    BE IN THE NEW
                   *   PLASMA DISPLAY CENTER.
                   *
                   * PARAMETERS:
                   *   - INDEX.- INDICATES THE RELATIVE POSITION OF THE CONTACT
                -   IN THE CONTACT$-
                   *     INFO STRUCTURE.
                   *
                   *************************************************************
                -   ***************/
 407   1           PUT$CONTACT$CENTER: PROCEDURE (INDEX) PUBLIC;
 408   2              DCL (POINTER, INDEX) BYTE;
 409   2              POINTER = CONTACT$INFO(INDEX).POINTER;
 410   2              CALL FSUB(.CONTACT$POSI(POINTER).X, .HALF$WINDOW, .ORIGIN$
                -   X);
 411   2              CALL FADD(.CONTACT$POSI(POINTER).Y, .HALF$WINDOW, .ORIGIN$
                -   Y);
 412   2              END PUT$CONTACT$CENTER;
```

```
                   /*************************************************************
             -      ***************
                   *
                   * FIXED$REORIENTATION:
                   *   THIS PROCEDURE IS USED TO DEFINE A NEW REFERENCE POINT FO
             -      R THE PLASMA
                   *   DISPLAY, ACCORDING TO 8 PREDEFINED POINTS.
                   *
                   * PARAMETERS:
                   *   - TYPE.- BYTE NUMBER BETWEEN 0 AND 7 THAT IS USED TO IDEN
             -      TIFY THE 8 PRE-
                   *     DEFINED WAYS TO REORIENT THE PLASMA SCREEN.
                   *
                   *************************************************************
             -      *****************/
 413   1           FIXED$REORIENTATION: PROCEDURE (TYPE) PUBLIC;
 414   2             DCL TYPE BYTE;
 415   2             DO CASE TYPE;
 416   3               DO;        /* CASE 0 */
 417   4                 CALL FADD(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
 418   4               END;

 419   3               DO;        /* CASE 1 */
 420   4                 CALL FADD(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
 421   4                 CALL FADD(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
 422   4               END;

 423   3               DO;        /* CASE 2 */
 424   4                 CALL FADD(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
 425   4          END;

 426   3          DO;        /* CASE 3 */
 427   4            CALL FADD(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
 428   4            CALL FSUB(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
 429   4          END;

 430   3          DO;        /* CASE 4 */
 431   4            CALL FSUB(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
 432   4          END;

 433   3          DO;        /* CASE 5 */
 434   4            CALL FSUB(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
 435   4            CALL FSUB(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
 436   4          END;

 437   3          DO;        /* CASE 6 */
 438   4            CALL FSUB (.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
 439   4          END;

 440   3          DO;        /* CASE 7 */
 441   4            CALL FSUB(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
 442   4            CALL FADD(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
 443   4          END;

 444   3          END;      /* END CASE */
 445   2        END FIXED$REORIENTATION;
```

```
                 /***************************************************************
             -    ***************
                 *
                 * PLASMA$REDESIG:
                 *   THIS PROCEDURE IS USED TO DISPLAY, IF POSSIBLE, THE NEW C
             -    ONTACT'S DESIG
                 *   IN THE LAST KNOWN POSITION.
                 *
                 * PARAMETERS:
                 *   - INDEX.- INDICATES THE RELATIVE POSITION OF THE CONTACT
             -    IN THE CONTACT$-
                 *     INFO STRUCTURE.
                 *
                 ****************************************************************
             -    ***************/
 446   1         PLASMA$REDESIG: PROCEDURE (INDEX) PUBLIC;
 447   2           DCL (X, Y) ADDRESS,
                       INDEX BYTE;
 448   2           IF LAST$POSI(INDEX).FLAG
                   THEN DO;
 450   3                X = LAST$POSI(INDEX).X;
 451   3                Y = LAST$POSI(INDEX).Y;
 452   3                CALL GRAPHIC$DESIG(X, Y, .CONTACT$INFO(INDEX).DESIG);
 453   3                END;
 454   2           END PLASMA$REDESIG;


                 /***************************************************************
             -    ***************
```

```
                 *
                 * PLASMA$DELETE:
                 *   THISPROCEDURE IS USED TO CLEAR THE FLAG CORRESPONDING TO
               -  A CONTACT THAT
                 *   HAS BEEN REMOVED FROM THE SYSTEM.
                 *
                 * PARAMETERS:
                 *   - INDEX.- INDICATES THE RELATIVE POSITION OF THE CONTACT
               -  IN THE CONTACTS-
                 *     INFO STRUCTURE.
                 *
                 ****************************************************************
               -  ***************/
 455   1         PLASMA$DELETE: PROCEDURE (INDEX) PUBLIC;
 456   2            DCL INDEX BYTE;
 457   2            LAST$POSI(INDEX).FLAG = FALSE;
 458   2            END PLASMA$DELETE;


                 /****************************************************************
               -  ***************
                 *
                 * PLASMA$CONTACT:
                 *   THIS PROCEDURE IS USED TO TRY TO PLOT THE LAST POSITION O
               -  F A GIVEN CONT-
                 *   ACT, IF POSSIBLE.
                 *
                 * PARAMETERS:
                 *   - INDEX.- INDICATES THE RELATIVE POSITION OF THE CONTACT
```

```
            -   IN THE CONTACT$-
            =       INFO STRUCTURE.
            *
            ***************************************************************
            -   ***************/
 459    1     PLASMA$CONTACT: PROCEDURE (INDEX) PUBLIC;
 460    2       DCL (X, Y) ADDRESS,
                    (POINTER, INDEX, TEMP) BYTE;
 461    2       POINTER = CONTACT$INFO(INDEX).POINTER;
 462    2       TEMP = CHECK$PLASMA(.CONTACT$POSI(POINTER).X, .CONTACT$POS
            -   I(POINTER).Y);
 463    2       IF LAST$POSI(INDEX).FLAG
                THEN DO;
 465    3           IF TEMP
                    THEN DO;
 467    4               CALL START$VECTOR$DASH(LAST$POSI(INDEX).X, LAST$
            -   POSI(INDEX).Y);
 468    4               CALL NORMALIZE(.CONTACT$POSI(POINTER).X, .CONTAC
            -   T$POSI(POINTER).Y,
                                          .X, .Y);
 469    4               CALL STOP$VECTOR$DASH(X, Y);
 470    4               IF CONTACT$INFO(INDEX).KIND = 0
                           THEN CALL DRAW$FRIENDLY$SYMBOL(X, Y);
                           ELSE CALL DRAW$UNK$HOS$SYMBOL(X, Y);
 472    4               LAST$POSI(INDEX).X = X;
 473    4               LAST$POSI(INDEX).Y = Y;
 474    4               END;
 475    4           ELSE DO;
 476    3               LAST$POSI(INDEX).FLAG = FALSE;
 477    4               END;
 478    4
 479   3              END;
 480   2          ELSE DO;
 481   3              IF TEMP
                      THEN DO;
 483   4                  CALL NORMALIZE(.CONTACT$POSI(POINTER).X, .CONTAC
           -   T$POSI(POINTER).Y,
                                      .X, .Y);
 484   4                  CALL GRAPHIC$DESIG(X, Y, .CONTACT$INFO(INDEX).DE
           -   SIG);
 485   4                  IF CONTACT$INFO(INDEX).KIND = 0
                             THEN CALL DRAW$FRIENDLY$SYMBOL(X, Y);
 487   4                     ELSE CALL DRAW$UNK$HOS$SYMBOL(X, Y);
 488   4                  LAST$POSI(INDEX).FLAG = TRUE;
 489   4                  LAST$POSI(INDEX).X = X;
 490   4                  LAST$POSI(INDEX).Y = Y;
 491   4                  END;
 492   3              END;
 493   2          END PLASMA$CONTACT;


               /************************************************************
           -    ****************
               *
               * PLASMA$OS:
               *   THIS PROCEDURE IS USED TO DISPLAY, IF POSSIBLE, THE LAST
           -    KNOWN POSITION
               *   OF THE OWN SHIP IN THE PLASMA DISPLAY.
               *
               *************************************************************
```

```
            -   ***************/
 494   1        PLASMA$OS: PROCEDURE PUBLIC;
 495   2          DCL (X, Y) ADDRESS,
                      (POINTER, TEMP) BYTE;
 496   2          POINTER = OWN$SHIP$INFO.POINTER;
 497   2          TEMP = CHECK$PLASMA(.OWN$SHIP(POINTER).X, .OWN$SHIP(POINTE
            -   R).Y);
 498   2          IF OS$LAST$POSI.FLAG
                  THEN DO;
 500   3              IF TEMP
                      THEN DO;
 502   4                  CALL START$VECTOR$SOLID(OS$LAST$POSI.X, OS$LAST$
            -   POSI.Y);
 503   4                  CALL NORMALIZE(.OWN$SHIP(POINTER).X, .OWN$SHIP(P
            -   OINTER).Y,
                                            .X, .Y);
 504   4                  CALL STOP$VECTOR$SOLID(X, Y);
 505   4                  CALL DRAW$OWN$SHIP$SYMBOL(X, Y);
 506   4                  OS$LAST$POSI.X = X;
 507   4                  OS$LAST$POSI.Y = Y;
 508   4                  END;
 509   3              ELSE DO;
 510   4                  OS$LAST$POSI.FLAG = FALSE;
 511   4                  END;
 512   3              END;
 513   2          ELSE DO;
 514   3              IF TEMP
                      THEN DO;
 516   4                  CALL NORMALIZE(.OWN$SHIP(POINTER).X, .OWN$SHIP(P
            -   OINTER).Y,
                                         .I, .Y);
 517   4                       CALL DRAW$OWN$SHIP$SYMBOL(X, Y);
 518   4                       OS$LAST$POSI.FLAG = TRUE;
 519   4                       OS$LAST$POSI.X = X;
 520   4                       OS$LAST$POSI.Y = Y;
 521   4                       END;
 522   3                   END;
 523   2               END PLASMA$OS;


                   /************************************************************
              -     ***************
                   *
                   * DRAW$EVERYTHING:
                   *   THIS PROCEDURE IS USED TO REDRAW THE ENTIRE DISPLAY.
                   *
                   *************************************************************
              -     *************/
 524   1           DRAW$EVERYTHING: PROCEDURE PUBLIC;
 525   2               DCL (POINTER, I, J ,P, TEMP) BYTE;
 526   2               CALL CLEAR$PLASMA;
 527   2               POINTER = OWN$SHIP$INFO.POINTER;
 528   2               IF OWN$SHIP$INFO.FLAG
                       THEN DO;
 530   3                   P = POINTER + 1;
 531   3                   DO I = 0 TO 29;
 532   4                       IF P >= 30 THEN P = 0;
 534   4                       OWN$SHIP$INFO.POINTER = P;
 535   4                       CALL PLASMA$OS;
 556   4              P = P + 1;
 537   4              END;
 538   3          END;
 539   2      ELSE DO;
 540   3          DO I = 0 TO POINTER;
 541   4              OWN$SHIP$INFO.POINTER = I;
 542   4              CALL PLASMA$OS;
 543   4              END;
 544   3          END;
 545   2      OWN$SHIP$INFO.POINTER = POINTER;
 546   2      DO I = 0 TO 14;
 547   3         IF CONTACT$INFO(I).DESIG <> 00H
               THEN DO;
 549   4              POINTER = CONTACT$INFO(I).POINTER;
 550   4              TEMP = CONTACT$INFO(I).POINTER / 15;
 551   4              TEMP = (TEMP + 1) * 15;
 552   4              IF CONTACT$INFO(I).FLAG
                      THEN DO;
 554   5                   P = POINTER + 1;
 555   5                   DO J = 0 TO 14;
 556   6                      IF P >= TEMP THEN P = (POINTER / 15) * 15;
 558   6                      CONTACT$INFO(I).POINTER = P;
 559   6                      CALL PLASMA$CONTACT(I);
 560   6                      P = P + 1;
 561   6                      END;
 562   5                   END;
 563   4              ELSE DO;
 564   5                   DO J = ((POINTER / 15) * 15) TO POINTER;
 565   6                      CONTACT$INFO(I).POINTER = J;
 566   6                      CALL PLASMA$CONTACT(I);
```

```
567   6                      END;
568   5                    END;
569   4                CONTACT$INFO(1).POINTER = POINTER;
570   4                END;
571   3            END;
572   2         END DRAW$EVERYTHING;


                /*************************************************************
            -    **************
                *
                * DISPLAY$PLASMA$SCALE:
                *   THIS PROCEDURE IS USED TO DISPLAY AT THE PLASMA DISPLAY,
            -    THE SCALE BEING
                *   USED TO DRAW THE PICTURE.
                *
                *************************************************************
            -    ***************/
573   1         DISPLAY$PLASMA$SCALE: PROCEDURE PUBLIC;
574   2           DCL TEMP BYTE;
575   2           TEMP = FP$FORMAT(.SYSTEM.SCALE, .BUFFER(7), 2, 2);
576   2           BUFFER(12), BUFFER(13) = EOL;
577   2           BUFFER(11) = BUFFER(10);
578   2           BUFFER(10) = BUFFER(9);
579   2           BUFFER(9) = POINT;
580   2           CALL PLASMA$PRINT$STRING(0, 2, .BUFFER);
581   2           END DISPLAY$PLASMA$SCALE;
```

```
                  /************************************************************
               -   *************
                  *
                  * REORIENT$PS:
                  *   THIS PROCEDURE IS USED TO DEFINE A NEW REFERENCE POINT FO
               -   R THE PICTURE
                  *   TO BE PRESENTED AT THE PLASMA DISPLAY.
                  *
                  ************************************************************
               -   *************/
 582   1          REORIENT$PS: PROCEDURE PUBLIC;
 583   2             DCL DESIG ADDRESS,
                         (TYPE, OK, REORIENT, INDEX, TEMP, FLAG, COUNT) BYTE;
 584   2             DCL TITLE (*) BYTE DATA
                               ('                                 PICTURE REORIENTA
               -   TION.$$'),
                         MSG0 (*) BYTE DATA
                               ('IF FIXED REORIENTATION IS DESIRED, TYPE 0.$$'),
                         MSG1 (*) BYTE DATA
                               ('IF OWN SHIP AT CENTER IS DESIRED, TYPE 1.$$'),
                         MSG2 (*) BYTE DATA
                               ('IF A CONTACT IS DESIRED AT CENTER, TYPE 2.$$'),
                         MSG3 (*) BYTE DATA
                               ('ENTER VALUE:  $$'),
                         MSG4 (*) BYTE DATA
                               ('ENTER POINT DESIRED TO BE AT CENTER:  $$'),
                         MSG5 (*) BYTE DATA
                               ('ENTER CONTACT DESIG AS REQUESTED:$$'),
                         MSG6 (*) BYTE DATA
```

```
                                ('DESIG NOT IN USE.$$');

 585   2          IF SYSTEM.NUMCTS > 0
                  THEN DO;
 587   3               FLAG = TRUE;
 588   3               COUNT = '2';
 589   3               END;
 590   2          ELSE DO;
 591   3               FLAG = FALSE;
 592   3               COUNT = '1';
 593   3               END;
 594   2          CALL CLEAR$STRUCTURES;
 595   2          CALL SET$WINDOW;
 596   2          OK = 0;
 597   2          DO WHILE OK = 0;
 598   3             CALL CRT$PRINT$STRING(.TITLE);
 599   3             CALL SEND$CRLF;
 600   3             CALL CRT$PRINT$STRING(.MSG0);
 601   3             CALL SEND$CRLF;
 602   3             CALL CRT$PRINT$STRING(.MSG1);
 603   3             CALL SEND$CRLF;
 604   3             IF FLAG
                      THEN DO;
 606   4                CALL CRT$PRINT$STRING(.MSG2);
 607   4                CALL SEND$CRLF;
 608   4                END;
 609   3             CALL CRT$PRINT$STRING(.MSG5);
 610   3             REORIENT = CRT$READ;
 611   3             DO WHILE (REORIENT < '0') OR (REORIENT > COUNT);
 612   4                CALL SEND$BEL;
 613   4                REORIENT = CRT$READ;
 614   4                END;
 615   3             CALL CRT$WRITE(REORIENT);
 616   3             CALL SEND$CRLF;
 617   3             OK = CHECK$INPUT;
 618   3             CALL CLEAR$LOW$SCREEN;
 619   3             END;
 620   2          REORIENT = REORIENT - 30H;
 621   2          DO CASE REORIENT;

 622   3             DO;       /* CASE 0. FIXED REORIENTATION. */
 623   4                OK = 0;
 624   4                DO WHILE OK = 0;
 625   5                   CALL CRT$PRINT$STRING(.TITLE);
 626   5                   CALL SEND$CRLF;
 627   5                   CALL CRT$PRINT$STRING(.MSG4);
 628   5                   TYPE = CRT$READ;
 629   5                   DO WHILE (TYPE < '0') OR (TYPE > '7');
 630   6                      CALL SEND$BEL;
 631   6                      TYPE = CRT$READ;
 632   6                      END;
 633   5                   CALL CRT$WRITE(TYPE);
 634   5                   CALL SEND$CRLF;
 635   5                   OK = CHECK$INPUT;
 636   5                   CALL CLEAR$LOW$SCREEN;
 637   5                   END;
 638   4                CALL FIXED$REORIENTATION(TYPE - 30H);
 639   4             END;
 640   3             DO;       /* CASE 1. OWN SHIP AT CENTER. */
 641   4                CALL PUT$OS$CENTER;
```

```
642   4              END;
643   3              DO;      /* CASE 2. CONTACT AT CENTER. */
644   4                 OK = 0FFH;
645   4                 DO WHILE OK = 0FFH;
646   5                    CALL CRT$PRINT$STRING(.TITLE);
647   5                    CALL SEND$CRLF;
648   5                    CALL CRT$PRINT$STRING(.MSG5);
649   5                    CALL SEND$CRLF;
650   5                    DESIG = GET$DESIG;
651   5                    INDEX = CHECK$DESIG(DESIG);
652   5                    IF INDEX = 0FFH
                           THEN DO;
654   6                         CALL CRT$PRINT$STRING(.MSG6);
655   6                         CALL SEND$CRLF;
656   6                         CALL SEND$CRLF;
657   6                         CALL CHECK$GO$KEY;
658   6                         END;
659   5                    OK = INDEX;
660   5                    CALL CLEAR$LOW$SCREEN;
661   5                    END;
662   4                 CALL PUT$CONTACT$CENTER(INDEX);
663   4              END;

664   3              END;      /* END CASE */

665   2           CALL DRAW$EVERYTHING;
666   2           CALL DISPLAY$PLASMA$SCALE;
667   2           END REORIENT$PS;
```

```
 668   1      END PLASMA$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 0DABH   3499D
     VARIABLE AREA SIZE = 00CFH    207D
     MAXIMUM STACK SIZE = 000CH     12D
     1323 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER                                        20 MAR 81   PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PLASMAPRIMITIVES
OBJECT MODULE PLACED IN PSPRIM.OBJ
COMPILER INVOKED BY:  :F1:PLM80 PSPRIM.SRC PAGELENGTH(33) PAGEWIDTH(75) DAT
                     -E(20 MAR 81)



  1            PLASMA$PRIMITIVES: DO;




  2   1        DECLARE LIT LITERALLY 'LITERALLY',
                     DCL LIT 'DECLARE';

  3   1        DCL TRUE LIT '0FFH',
                   FALSE LIT '00H';

  4   1        DCL PLASMA$DATA       LIT '04H',
                   PLASMA$STATUS     LIT '05H',
                   RECEIVE$MASK      LIT '06H',
                   TRANSMIT$MASK     LIT '05H';

  5   1        DCL STATUS$A          LIT '04H',
                   RESET$ALL         LIT '00H',
                   OUT$BUSY          LIT '01H',
                   IN$BUSY           LIT '02H';
  6   1       DCL STX                 LIT '02H',    /* START TEXT */
                  ETX                 LIT '03H',    /* ENABLE TEXT */
                  CS                  LIT '0CH',    /* CLEAR SCREEN */
                  CG                  LIT '0EH',    /* CONSTRUCT GRAPH */
                  CV                  LIT '0FH',    /* CLEAR VECTORS */
                  EOL                 LIT '24H';    /* END OF LINE */

  7   1       DCL SET$ERASE           LIT '0100$0000B',
                  SET$DASHED          LIT '0001$0000B',
                  SET$END             LIT '0010$0000B';


              /*************************************************************
            -  ***********
              *
              * SET$STATUS$PLASMA:
              *   THIS PROCEDURE IS USED TO SET THE STATUS LINE FOR THE PLA
            -  SMA.
              *   NOTE THAT THE LOGIC TO BE USED IS NEGATIVE.
              *
              * PARAMETERS:
              *   - STATUS.- ASCII CHARACTER USED TO DEFINE THE STATUS LINE
            -  .
              *
              *************************************************************
            -  **********/
  8   1      SET$STATUS$PLASMA: PROCEDURE (STATUS) PUBLIC;
  9   2         DCL STATUS BYTE;
 10   2         OUTPUT(PLASMA$STATUS) = NOT STATUS;
 11   2         END SET$STATUS$PLASMA;


             /************************************************************
         -    ************
             *
             * PLASMA$WRITE:
             *   THIS PROCEDURE IS USED TO SEND A CHARACTER TO THE PLASMA
         -    DISPLAY.
             *
             * PARAMETERS:
             *   - CHAR.- ASCII CHARACTER DESIRED TO BE SENT.
             *
             *************************************************************
         -    ************/
 12   1      PLASMA$WRITE: PROCEDURE (CHAR) PUBLIC;
 13   2         DCL CHAR BYTE;
 14   2         DO WHILE ((NOT INPUT(PLASMA$STATUS)) AND STATUS$A) <> STAT
         -    US$A;
 15   3            END;        /* WAIT FOR PLASMA TO BE READY */
 16   2         CALL SET$STATUS$PLASMA(RESET$ALL);
 17   2         OUTPUT(PLASMA$DATA) = NOT CHAR;
 18   2         CALL SET$STATUS$PLASMA(OUT$BUSY);
 19   2         END PLASMA$WRITE;
```

```
                /************************************************************
          -      *************
                *
                * CLEAR$PLASMA:
                *   THIS PROCEDURE IS USED TO CLEAR THE PLASMA DISPLAY.
                *
                *************************************************************
          -      *************/
  20   1        CLEAR$PLASMA: PROCEDURE PUBLIC;
  21   2           CALL SET$STATUS$PLASMA(IN$BUSY);
  22   2           CALL SET$STATUS$PLASMA(RESET$ALL);
  23   2           CALL PLASMA$WRITE(CS);
  24   2           CALL PLASMA$WRITE(CV);
  25   2           CALL SET$STATUS$PLASMA(RESET$ALL);
  26   2           END CLEAR$PLASMA;


                /************************************************************
          -      **************
                *
                * PLASMA$WRITE$VECTOR:
                *   THIS PROCEDURE IS USED TO SEND A FOUR BYTE VECTOR TO THE
          -      PLASMA.
                *
                * PARAMETERS:
                *   - A.- POINTER TO THE FOUR BYTE VECTOR DESIRED TO BE SENT.
                *
                *************************************************************
```

```
PL/M-80 COMPILER                                                      20 MAR 81  PAGE   5

               -   *************/
  27   1          PLASMA$WRITE$VECTOR: PROCEDURE (A) PUBLIC;
  28   2             DCL A ADDRESS,
                         VECTOR BASED A (4) BYTE,
                         VPTR BYTE;
  29   2             DO VPTR = 0 TO 3;
  30   3                CALL PLASMA$WRITE(VECTOR(VPTR));
  31   3                END;
  32   2             END PLASMA$WRITE$VECTOR;


                  /*************************************************************
               -   *************
                  *
                  * PLASMA$PRINT$STRING:
                  *   THIS PROCEDURE IS USED TO WRITE A GIVEN STRING IN A GIVEN
               -   POSITION AT
                  *   THE PLASMA DISPLAY.
                  *
                  * PARAMETERS:
                  *    - COLUMN.- DENOTES THE COLUMN NUMBER DESIRED TO BE ADDRES
               -   SED.
                  *    - ROW.- DENOTES THE ROW NUMBER DESIRED TO BE ADDRESSED.
                  *    - POINTER.- POINTS TO THE FIRST BYTE OF THE STRING DESIRE
               -   D TO BE DIS-
                  *      PLAYED. NOTE THAT TWO CONSECUTIVE '$' SIGNS MUST MARK T
               -   HE END OF
                  *      THE STRING.
                  *
                  ************************************************************
               -   ***************/
 33   1           PLASMA$PRINT$STRING: PROCEDURE (COLUMN, ROW, POINTER) PUBLIC;
 34   2              DCL POINTER ADDRESS,
                         BUFFER BASED POINTER (1) BYTE,
                         (COLUMN, ROW, COUNT) BYTE;
 35   2              COUNT = 0;
 36   2              CALL SET$STATUS$PLASMA(IN$BUSY);
 37   2              CALL SET$STATUS$PLASMA(RESET$ALL);
 38   2              CALL PLASMA$WRITE(STX);
 39   2              CALL PLASMA$WRITE(COLUMN);
 40   2              CALL PLASMA$WRITE(ROW);
 41   2              DO WHILE (BUFFER(COUNT) <> EOL ) OR (BUFFER(COUNT + 1) <>
               -   EOL );
 42   3                 CALL PLASMA$WRITE(BUFFER(COUNT));
 43   3                 COUNT = COUNT + 1;
 44   3                 END;
 45   2              CALL PLASMA$WRITE(ETX);
 46   2              CALL SET$STATUS$PLASMA(RESET$ALL);
 47   2              END PLASMA$PRINT$STRING;


                  /************************************************************
               -   ***************
                  *
                  * INITIALIZE PLASMA:
                  *   THIS PROCEDURE IS USED TO INITIALIZE THE PLASMA DISPLAY.
                  *
                  ************************************************************
```

```
              -   **************/
  48   1          INITIALIZE$PLASMA: PROCEDURE PUBLIC;
  49   2             DCL BUFFER (*) BYTE DATA ('ON LINE.$$');
  50   2             CALL CLEAR$PLASMA;
  51   2             CALL PLASMA$PRINT$STRING(0, 2, .BUFFER);
  52   2             END INITIALIZE$PLASMA;


                  /*************************************************************
              -   *************
                  *
                  * SET$VECTOR:
                  *   THIS PROCEDURE IS USED PREPARE TWO X AND Y VALUES GIVEN I
              -   NTO THE FOR-
                  *   MAT REQUIRED BY THE PLASMA UNIT TO DEFINE A VECTOR.
                  * PARAMETERS:
                  *   - X.- ADDRESS VALUE.
                  *   - Y.- ADDRESS VALUE.
                  *   - POINTER.- POINTS TO A FOUR BYTE VECTOR IN WHICH THE X A
              -   ND Y VALUES
                  *     WILL BE ARRANGED.
                  *
                  **************************************************************
              -   ************/
  53   1          SET$VECTOR: PROCEDURE (X, Y, POINTER) PUBLIC;
  54   2             DCL (X, Y, POINTER) ADDRESS,
                         VECTOR BASED POINTER (4) BYTE;
  55   2             VECTOR(0) = CG;
  56   2             VECTOR(1), VECTOR(2), VECTOR(3) = 00H;
```

```
 57   2        VECTOR(1) = LOW(X) AND 07FH;
 58   2        VECTOR(2) = LOW(Y) AND 07FH;
 59   2        VECTOR(3) = HIGH(SHL(Y AND 180H, 3)) OR
                           HIGH(SHL(X AND 180H, 1));
 60   2        VECTOR(3) = VECTOR(3) AND NOT SET$ERASE;
 61   2        END SET$VECTOR;


             /**************************************************************
          -   ***************
             *
             * START$VECTOR$SOLID:
             *    THIS PROCEDURE IS USED TO DEFINE A START POINT FOR A SOLI
          -   D VECTOR.
             *
             * PARAMETERS:
             *    - X.- ADDRESS VALUE.
             *    - Y.- ADDRESS VALUE.
             *
             ***************************************************************
          -   **************/
 62   1      START$VECTOR$SOLID: PROCEDURE (X, Y) PUBLIC;
 63   2         DCL (X, Y) ADDRESS,
                    VECTOR (4) BYTE;
 64   2         CALL SET$VECTOR(X, Y, .VECTOR);
 65   2         VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
 66   2         VECTOR(3) = VECTOR(3) AND NOT SET$END;
 67   2         CALL PLASMA$WRITE$VECTOR(.VECTOR);
 68   2         END START$VECTOR$SOLID;
```

```
                  /*************************************************************
               -   **************
                  *
                  * STOP$VECTOR$SOLID:
                  *   THIS PROCEDURE IS USED TO DEFINE A STOP POINT FOR A SOLID
               -    VECTOR.
                  *
                  * PARAMETERS:
                  *   - X.- ADDRESS VALUE.
                  *   - Y.- ADDRESS VALUE.
                  *
                  *************************************************************
               -   *************/
  69   1          STOP$VECTOR$SOLID: PROCEDURE (X, Y) PUBLIC;
  70   2             DCL (X, Y) ADDRESS,
                         VECTOR (4) BYTE;
  71   2             CALL SET$VECTOR(X, Y, .VECTOR);
  72   2             VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
  73   2             VECTOR(3) = VECTOR(3) OR SET$END;
  74   2             CALL PLASMA$WRITE$VECTOR(.VECTOR);
  75   2             END STOP$VECTOR$SOLID;


                  /*************************************************************
               -   **************
                  *
```

```
PL/M-80 COMPILER                                          20 MAR 81   PAGE  10

                 * START$VECTOR$DASH:
                 *   THIS PROCEDURE IS USED TO DEFINE A START POINT FOR A DASH
               -  ED VECTOR.
                 *
                 * PARAMETERS:
                 *   - X.- ADDRESS VALUE.
                 *   - Y.- ADDRESS VALUE.
                 *
                 ************************************************************
               -  **************/
  76   1         START$VECTOR$DASH: PROCEDURE (X, Y) PUBLIC;
  77   2            DCL (X, Y) ADDRESS,
                        VECTOR (4) BYTE;
  78   2            CALL SET$VECTOR(X, Y, .VECTOR);
  79   2            VECTOR(3) = VECTOR(3) OR SET$DASHED;
  80   2            VECTOR(3) = VECTOR(3) AND NOT SET$END;
  81   2            CALL PLASMA$WRITE$VECTOR(.VECTOR);
  82   2            END START$VECTOR$DASH;


                 /************************************************************
               -  **************
                 *
                 * STOP$VECTOR$DASH:
                 *   THIS PROCEDURE IS USED TO DEFINE A STOP POINT FOR A DASHE
               -  D VECTOR.
                 *
                 * PARAMETERS:
                 *   - X.- ADDRESS VALUE.
```

```
                   *   - Y.- ADDRESS VALUE.
                   *
                   ****************************************************************
                -   **************/
 83   1            STOP$VECTOR$DASH: PROCEDURE (X, Y) PUBLIC;
 84   2               DCL (X, Y) ADDRESS,
                          VECTOR (4) BYTE;
 85   2               CALL SET$VECTOR(X, Y, .VECTOR);
 86   2               VECTOR(3) = VECTOR(3) OR SET$DASHED;
 87   2               VECTOR(3) = VECTOR(3) OR SET$END;
 88   2               CALL PLASMA$WRITE$VECTOR(.VECTOR);
 89   2               END STOP$VECTOR$DASH;


                   /****************************************************************
                -   **************
                   *
                   * GRAPHIC$DESIG:
                   *   THIS PROCEDURE IS USED TO DISPLAY THE DESIGNATION OF A CO
                -   NTACT IN THE
                   *   NEAREST POSSIBLE ALPHANUMERIC LOCATION TO THE X, Y VALUES
                -    GIVEN.
                   *
                   * PARAMETERS:
                   *   - X.- ADDRESS VALUE.
                   *   - Y.- ADDRESS VALUE.
                   *   - DESIG.- POINTER TO THE DESIG OF A CONTACT.
                   *
                   ****************************************************************
                -   ***************/
 90   1       GRAPHIC$DESIG: PROCEDURE (X, Y, DESIG) PUBLIC;
 91   2          DCL (X, Y, DESIG) ADDRESS,
                     VALUE BASED DESIG ADDRESS,
                     BUFFER (6) BYTE,
                     (ROW, COLUMN) BYTE;
 92   2          BUFFER(0) = '[';
 93   2          BUFFER(1) = VALUE / 100;
 94   2          BUFFER(2) = VALUE MOD 100;
 95   2          BUFFER(3) = ']';
 96   2          BUFFER(4), BUFFER(5) = EOL;
 97   2          ROW = Y / 16;
 98   2          COLUMN = ((X - 14) / 6) - 5;
 99   2          IF X <= 8 THEN COLUMN = 0;
101   2          IF (X >= 9) AND (X <= 44) THEN COLUMN = X / 6;
103   2          IF (X >= 494) THEN COLUMN = 75;
105   2          CALL PLASMA$PRINT$STRING(COLUMN, ROW, .BUFFER);
106   2          END  GRAPHIC$DESIG;


107   1       END PLASMA$PRIMITIVES;


MODULE INFORMATION:

     CODE AREA SIZE     = 02E6H    742D
     VARIABLE AREA SIZE = 003EH     62D
     MAXIMUM STACK SIZE = 0006H      6D
```

```
PL/M-80 COMPILER                                              20 MAR 81  PAGE  13

     303 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER                                      20 MAR 81  PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE EXECUTIVECMDS
OBJECT MODULE PLACED IN :F1:EXEC2.OBJ
COMPILER INVOKED BY:  PLM80 :F1:EXEC2.SRC PAGELENGTH(33) PAGEWIDTH(75) DATE
                     -(20 MAR 81)


   1          EXECUTIVE$CMDS: DO;


              $NOLIST


                                          /***   EXTERNALS:   ***/


 201   1      DECLARE LIT LITERALLY 'LITERALLY',
                      DCL LIT 'DECLARE';


 202   1      GET$CPA:
                 PROCEDURE (A, B) BYTE EXTERNAL;
 203   2         DCL A BYTE, B ADDRESS; END;

 205   1      PLASMA$REDESIG:
                 PROCEDURE (INDEX) EXTERNAL;
 206   2         DCL INDEX BYTE; END;
```

```
208   1      PLASMA$DELETE:
                PROCEDURE (INDEX) EXTERNAL;
209   2         DCL INDEX BYTE; END;

211   1      PLASMA$CONTACT:
                PROCEDURE (INDEX) EXTERNAL;
212   2         DCL INDEX BYTE; END;

214   1      CLEAR$STRUCTURES:
                PROCEDURE EXTERNAL;
215   2         END;

216   1      SET$WINDOW:
                PROCEDURE EXTERNAL;
217   2         END;

218   1      PUT$OS$CENTER:
                PROCEDURE EXTERNAL;
219   2         END;

220   1      DRAW$EVERYTHING:
                PROCEDURE EXTERNAL;
221   2         END;

222   1      PLASMA$OS:
                PROCEDURE EXTERNAL;
223   2         END;

224   1      DISPLAY$PLASMA$SCALE:
                PROCEDURE EXTERNAL;
```

```
 225   2          END;

 226   1        MOVE$OWN$SHIP:
                  PROCEDURE EXTERNAL;
 227   2          END;

                                      /***   DECLARATIONS:   ***/



 228   1        DCL SAFE$RNG (4) BYTE PUBLIC INITIAL (08H, 3CH, 0CAH, 03CH);
                          /* 0.0246868270 MILES OR FIFTY YARDS */


 229   1        DCL SYSTEM STRUCTURE
                       (LAT (4) BYTE,
                        LONG (4) BYTE,
                        SCALE (4) BYTE,
                        WIND$DIR (4) BYTE,
                        WIND$SPD (4) BYTE,
                        NUM$ZONE (5) BYTE,
                        CONTACT$KIND (3) BYTE,
                        NUMCTS BYTE ) PUBLIC,

                  OWN$SHIP$INFO STRUCTURE
                            (LAT (4) BYTE,
                             LONG (4) BYTE,
                             POINTER BYTE,
                         FLAG BYTE) PUBLIC,

                OWN$SHIP (30) STRUCTURE
                        (X (4) BYTE,
                         Y (4) BYTE,
                         TIME (3) BYTE,
                         CRS (4) BYTE,
                         SPD (4) BYTE) PUBLIC,

                CONTACT$INFO (15) STRUCTURE
                           (DESIG ADDRESS,
                            TYPE BYTE,
                            KIND BYTE,
                            CRS$FLAG BYTE,
                            SPD$FLAG BYTE,
                            OS$POINTER BYTE,
                            POINTER BYTE,
                            FLAG BYTE) PUBLIC,

                CONTACT$POSI (225) STRUCTURE
                           (X (4) BYTE,
                            Y (4) BYTE,
                            TIME (3) BYTE,
                            CRS (4) BYTE,
                            SPD (4) BYTE,
                            BRG (4) BYTE,
                            RNG (4) BYTE) PUBLIC;

 230   1      DCL CONTACT$DISPLAY (10) BYTE PUBLIC;
 231   1       DCL LAT$STRING (9) BYTE PUBLIC,
                   LONG$STRING (9) BYTE PUBLIC,
                   CRS$STRING (6) BYTE PUBLIC,
                   SPD$STRING (5) BYTE PUBLIC,
                   CONTACTS$STRING (8) BYTE PUBLIC,
                   CONTACT$INFO$STRING (46) BYTE PUBLIC,
                   INPUT$MODE (*) BYTE DATA (' INPUT $$'),
                   DISPLAY$MODE (*) BYTE DATA ('DISPLAY$$');

 232   1       DCL FP$MINS$TO$RAD (4) BYTE DATA (98H,82H,98H,39H),     /* 2.00
           -    029089 */
                   FP$2 (4) BYTE DATA (00H,00H,00H,40H);               /* 2.0
           -    */

 233   1       DCL PROMPT LIT '25H';         /* PROMPT CHARACTER */

 234   1       DCL
                   MSG$0 (*) BYTE DATA ('PRESS THE ''GO'' KEY TO CONTINUE:$$
           -    '),
                   MSG$1 (*) BYTE DATA ('DO YOU NEED TO UPDATE$$'),
                   MSG$2 (*) BYTE DATA ('COURSE?    (Y/N) $$'),
                   MSG$3 (*) BYTE DATA ('SPEED?     (Y/N) $$'),
                   MSG$4 (*) BYTE DATA ('LATITUDE?  (Y/N) $$'),
                   MSG$5 (*) BYTE DATA ('LONGITUDE? (Y/N) $$'),
                   MSG$6 (*) BYTE DATA ('BEARING?   (Y/N) $$'),
                   MSG$7 (*) BYTE DATA ('RANGE?     (Y/N) $$'),
                   MSG$8 (*) BYTE DATA ('DESIG?     (Y/N) $$'),
                   MSG$9 (*) BYTE DATA ('TYPE?      (Y/N) $$'),
                    MSG$A (*) BYTE DATA ('CLASS?      (Y/N) $$'),
                    BLANK (*) BYTE DATA ('                    $$');

235   1       DCL
                    TITLE$0 (*) BYTE DATA
                            ('                    NEW CONTACT INITIALIZ
          -       ATION.$$'),
                    TITLE$1 (*) BYTE DATA
                            ('                       CONTACT REMOVAL.
          -       $$'),
                    TITLE$2 (*) BYTE DATA
                            ('                   CONTACT REDESIGNATIO
          -       N.$$'),
                    TITLE$3 (*) BYTE DATA
                            ('                        CONTACT UPDATE.
          -       $$'),
                    TITLE$4 (*) BYTE DATA
                            ('                   OWN SHIP DATA UPDAT
          -       ING.$$'),
                    TITLE$5 (*) BYTE DATA
                            ('                CHANGE OF CONTACTS BEING
          -       DISPLAYED.$$'),
                    TITLE$6 (*) BYTE DATA
                            ('                      CHANGE OF TIME PARA
          -       METERS.$$'),
                    TITLE$9 (*) BYTE DATA
                            ('                COORDINATE GRID ORIGIN MO
          -       DIFICATION.$$'),
                    TITLE$A (*) BYTE DATA
```

```
                 ('                                      CHANGE OF WIND INFORM
-   ATION.$$'),
       TITLE$B (*) BYTE DATA
                 ('                                      GRAPHICS SCALE MODIFI
-   CATION.$$'),
       TITLE$C (*) BYTE DATA
                 ('                                        PICTURE REORIENTAT
-   ION.$$'),
       TITLE$D (*) BYTE DATA
                 ('                                      CHANGE OF SAFE C.P.A.
-    RANGE$$');


   /*****************************************************************
-   ***************
   *
   * DE$HASH:
   *   THIS PROCEDURE IS USED TO GET A PAIR OF ASCII CHARACTERS,
-    REPRESENTING
   *   A CONTACT'S DESIGNATION, FROM AN ADDRESS VALUE. SEE 'GETS
-   DESIG'.
   *
   * PARAMETERS:
   *   - A.- ADDRESS VALUE CONTAINING THE CODE TO BE 'DEHASHED'
-   INTO TWO
   *     ASCII CHARACTERS.
   *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS
-    DESIRED TO BE
   *     PLACED.
```

```
                  *
                  **************************************************************
              -    ****************/
236    1          DE$HASH: PROCEDURE (A,B) PUBLIC;
237    2             DCL (A,B) ADDRESS,
                         CHAR BASED B BYTE;
238    2             CHAR = A / 100;
239    2             B = B + 1;
240    2             CHAR = A MOD 100;
241    2             END DE$HASH;


                  /*************************************************************
              -    ***************
                  *
                  * CHECK$GO$KEY:
                  *   THIS PROCEDURE IS USED TO CHECK IF THE 'GO' KEY IS PRESSE
              -    D.
                  *
                  **************************************************************
              -    *****************/
242    1          CHECK$GO$KEY: PROCEDURE PUBLIC;
243    2             DCL CHAR BYTE;
244    2             CALL CRT$PRINT$STRING(.MSG$0);
245    2             CALL SEND$BEL;
246    2             CHAR = CRT$READ;
247    2             DO WHILE CHAR <> 02CH;
248    3                CALL SEND$BEL;
249    3                CHAR = CRT$READ;
```

```
 250   3            END;
 251   2          END CHECK$GO$KEY;


                /*************************************************************
           -     ****************
                *
                * DISPLAY$KIND:
                *   THIS PROCEDURE IS USED TO DISPLAY THE INFORMATION ABOUT T
           -     HE KIND OF
                *   CONTACTS MAINTAINED BY THE SYSTEM.
                *
                *************************************************************
           -     *****************/
 252   1        DISPLAY$KIND: PROCEDURE PUBLIC;
 253   2          DCL I BYTE;
 254   2          DO I = 0 TO 2;
 255   3            CONTACTS$STRING(2 * I) = SYSTEM.CONTACT$KIND(I) / 10 +
           -     30H;
 256   3            CONTACTS$STRING((2 * I) + 1) = SYSTEM.CONTACT$KIND(I) M
           -     OD 10 + 30H;
 257   3            END;
 258   2          CONTACTS$STRING(6), CONTACTS$STRING(7) = '$';
 259   2          CALL PRINT$CONTACTS(.CONTACTS$STRING);
 260   2          END DISPLAY$KIND;


                /*************************************************************
```

```
          -   *****************
              *
              * CHECK$DESIG:
              *   THIS PROCEDURE IS USED TO USED TO DETECT THE PRESENCE OF
          -   A GIVEN CONTACT
              *   IN THE SYSTEM.
              *
              * PARAMETERS:
              *   - A.- ADDRESS VARIABLE THAT CONTAINS THE 'HASHED' VALUE O
          -   F THE CONTACT'S
              *     DESIGNATION DESIRED TO BE CHECKED.
              *
              * USAGE:
              *   TYPED PROCEDURE. A VALUE INDICATING THE RELATIVE POSITION
          -    OF THE CONTACT
              *   IS RETURNED IF FOUND. OTHERWISE A VALUE OF 0FFH IS RETURN
          -   ED.
              *
              ***************************************************************
          -   *****************/
 261   1      CHECK$DESIG: PROCEDURE (A) BYTE PUBLIC;
 262   2        DCL A ADDRESS,
                      I BYTE;
 263   2        DO I = 0 TO 14;
 264   3          IF CONTACT$INFO(I).DESIG = A THEN RETURN I;
 266   3          END;
 267   2        RETURN 0FFH;
 268   2        END CHECK$DESIG;
```

```
                 /*************************************************************
               -  *****************
                  *
                  * CONV$MIN$RAD:
                  *   THIS PROCEDURE IS USED TO CONVERT A GIVEN ANGLE, IN MINUT
               -  ES, TO RADIANS.
                  *
                  * PARAMETERS:
                  *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING
               -  POINT REPRESEN-
                  *     TATION OF AN ANGLE IN MINUTES, IS LOCATED.
                  *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE VALUE IN
               -  RADIANS IS DESIRED
                  *     TO BE PLACED.
                  *
                  *************************************************************
               -  ****************/
 269   1         CONV$MIN$RAD: PROCEDURE (A,B) PUBLIC;
 270   2            DCL (A,B) ADDRESS,
                        MIN BASED A (4) BYTE,
                        RAD BASED B (4) BYTE;
 271   2            CALL FMUL(.MIN,.FP$MIN$TO$RAD,.RAD);
 272   2            END CONV$MIN$RAD;


                 /*************************************************************
               -  *****************
                  *
                  * CONV$RAD$MIN:
                  *   THIS PROCEDURE IS USED TO CONVERT A GIVEN ANGLE, IN RADIA
               -  NS, TO MINUTES.
                  *
                  * PARAMETERS:
                  *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING
               -  POINT REPRESEN-
                  *     TATION OF AN ANGLE IN RADIANS, IS LOCATED.
                  *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE VALUE IN
               -  MINUTES IS DESI-
                  *     RED TO BE PLACED.
                  *
                  ***************************************************************
               -  ****************/
 273    1         CONV$RAD$MIN: PROCEDURE (A,B) PUBLIC;
 274    2           DCL (A,B) ADDRESS,
                        RAD BASED A (4) BYTE,
                        MIN BASED B (4) BYTE;
 275    2           CALL FDIV(.RAD,.FP$MIN$TO$RAD,.MIN);
 276    2           END CONV$RAD$MIN;



                  /**************************************************************
               -  ***************
                  *
                  * CONV$XY:
                  *   THIS PROCEDURE IS USED TO CONVERT GIVEN VALUES OF LATITUI
               -  E AND LONGITUDE
```

```
                 *   INTO 'X,Y' COORDINATES.
                 *
                 * PARAMETERS:
                 *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
               -  OF LATITUDE IS
                 *      LOCATED.
                 *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
               -  OF LONGITUDE IS
                 *      LOCATED.
                 *   - C.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
               -  REPRESENTING
                 *      'X' IS DESIRED TO BE PLACED.
                 *   - D.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
               -  REPRESENTING
                 *      'Y' IS DESIRED TO BE PLACED.
                 *
                 ****************************************************************
               -  ****************/
 277   1         CONV$XY: PROCEDURE (A,B,C,D) PUBLIC;
 278   2            DCL (A,B,C,D) ADDRESS,
                        LAT BASED A (4) BYTE,
                        LONG BASED B (4) BYTE,
                        X BASED C (4) BYTE,
                        Y BASED D (4) BYTE,
                        MEAN$LAT (4) BYTE,
                        COS$MEAN$LAT (4) BYTE,
                        SIN$MEAN$LAT (4) BYTE;
 279   2             CALL FADD(.SYSTEM.LAT, .LAT, .MEAN$LAT);
 280   2             CALL FDIV(.MEAN$LAT, .FP$2, .MEAN$LAT);
 281   2             CALL CONV$MIN$RAD(.MEAN$LAT, .MEAN$LAT);
 282   2          CALL COS$SIN(.MEAN$LAT, .COS$MEAN$LAT, .SIN$MEAN$LAT);
 283   2          CALL FSUB(.LONG, .SYSTEM.LONG, .X);
 284   2          CALL FMUL(.X, .COS$MEAN$LAT, .X);
 285   2          CALL FSUB(.LAT, .SYSTEM.LAT, .Y);
 286   2          END CONV$XY;


               /*****************************************************************
          -     **************
               *
               * CONV$REL$XY:
               *   THIS PROCEDURE IS USED TO OBTAIN VALUES OF 'X,Y' COORDINA
          -     TES FOR A POINT
               *   TO WHICH A RANGE AND BEARING ARE GIVEN.
               *
               * PARAMETERS:
               *   - A.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
          -     OF THE BEARING
               *     IS LOCATED.
               *   - B.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
          -     OF THE RANGE
               *     IS LOCATED.
               *   - C.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
          -     REPRESENTING
               *     'X' IS DESIRED TO BE PLACED.
               *   - D.- POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE
          -     REPRESENTING
               *     'Y' IS DESIRED TO BE PLACED.
```

```
                  *
                  ************************************************************
             -     ****************/
 287    1         CONV$REL$XY: PROCEDURE (A,B,C,D) PUBLIC;
 288    2            DCL (A,B,C,D) ADDRESS,
                         BRG BASED A (4) BYTE,
                         RNG BASED B (4) BYTE,
                         DELTA$X BASED C (4) BYTE,
                         DELTA$Y BASED D (4) BYTE,
                         COS (4) BYTE, SIN (4) BYTE,
                         ANGLE (4) BYTE,
                         (I, TEMP) BYTE;
 289    2            DCL DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH);  /*
             -     0.0174532925 */
 290    2            TEMP = OWN$SHIP$INFO.POINTER;
 291    2            DO I = 0 TO 3;
 292    3               ANGLE(I) = BRG(I);
 293    3               END;
 294    2            CALL FMUL(.ANGLE, .DEG$TO$RAD, .ANGLE);
 295    2            CALL COS$SIN(.ANGLE, .COS, .SIN);
 296    2            CALL FMUL(.RNG, .SIN, .DELTA$X);
 297    2            CALL FADD(.OWN$SHIP(TEMP).X, .DELTA$X, .DELTA$X);
 298    2            CALL FMUL(.RNG, .COS, .DELTA$Y);
 299    2            CALL FADD(.OWN$SHIP(TEMP).Y, .DELTA$Y, .DELTA$Y);
 300    2            END CONV$REL$XY;


                  /***********************************************************
             -     **************
```

```
                  *
                  * INIT$STRUCTURES:
                  *   THIS PROCEDURE IS USED TO INITIALIZE ALL STRUCTURES TO 0.
                  *
                  ****************************************************************
              -    ***************/
 301    1         INIT$STRUCTURES: PROCEDURE;
 302    2            DCL (A,I) ADDRESS,
                         TEMP BASED A BYTE;
 303    2            A = .SYSTEM;
 304    2            DO I = 0 TO 28;
 305    3               TEMP = 0;
 306    3               A = A + 1;
 307    3               END;
 308    2            A = .OWN$SHIP$INFO;
 309    2            DO I = 0 TO 9;
 310    3               TEMP = 0;
 311    3               A = A + 1;
 312    3               END;
 313    2            A = .OWN$SHIP;
 314    2            DO I = 0 TO 569;
 315    3               TEMP = 0;
 316    3               A = A + 1;
 317    3               END;
 318    2            A = .CONTACT$INFO;
 319    2            DO I = 0 TO 134;
 320    3               TEMP = 0;
 321    3               A = A + 1;
 322    3               END;
 323    2            A = .CONTACT$POSI;
 324   2        DO I = 0 TO 6074;
 325   3           TEMP = 0;
 326   3           A = A + 1;
 327   3           END;
 328   2        A = .CONTACT$DISPLAY;
 329   2        DO I = 0 TO 9;
 330   3           TEMP = 0FFH;
 331   3           A = A + 1;
 332   3           END;
 333   2        END INIT$STRUCTURES;

             /**************************************************************
           -  ***************
             *
             * GET$SYSTEM$PARAMETERS:
             *   THIS PROCEDURE IS USED BY THE EXECUTIVE TO INITIALIZE THE
           -  SYSTEM.
             *
             ***************************************************************
           -  **************/
 334   1     GET$SYSTEM$PARAMETERS: PROCEDURE PUBLIC;
 335   2        DCL MSG0 (*) BYTE DATA
                     ('                                SYSTEM INITIALIZATION$$
           -  '),
                     MSG1 (*) BYTE DATA ('TIME INITIALIZATION:$$'),
                     MSG2 (*) BYTE DATA ('COORDINATE GRID ORIGIN INITIALIZA
           -  TION:$$'),
                     MSG3 (*) BYTE DATA ('OWN SHIP INITIAL DATA:$$'),
                     MSG4 (*) BYTE DATA ('INITIAL GRAPHICS SCALE:$$'),
                      MSG5 (*) BYTE DATA ('PRESS THE ''GO'' KEY TO START THE
               -    SYSTEM.$$');

 336    2          DCL CHAR BYTE;

 337    2          CALL CRT$MASTER$CLEAR;
 338    2          CALL INIT$FP;
 339    2          CALL INIT$STRUCTURES;
 340    2          CALL INIT$HIGH$SCREEN;
 341    2          CALL SET$LOW$HOME;
 342    2          CALL CRT$PRINT$STRING(.MSG0);
 343    2          CALL SEND$CRLF;
 344    2          CALL CRT$PRINT$STRING(.MSG1);
 345    2          CALL SEND$CRLF;
 346    2          CALL GET$TIME$ZONE(.SYSTEM.NUM$ZONE);
 347    2          CALL CRT$PRINT$STRING(.MSG0);
 348    2          CALL SEND$CRLF;
 349    2          CALL CRT$PRINT$STRING(.MSG1);
 350    2          CALL SEND$CRLF;
 351    2          CALL INITIATE$TIME;
 352    2          CALL CLEAR$LOW$SCREEN;
 353    2          CALL CRT$PRINT$STRING(.MSG0);
 354    2          CALL SEND$CRLF;
 355    2          CALL CRT$PRINT$STRING(.MSG2);
 356    2          CALL SEND$CRLF;
 357    2          CALL GET$LAT(.SYSTEM.LAT);
 358    2          CALL CRT$PRINT$STRING(.MSG0);
 359    2          CALL SEND$CRLF;
 360    2          CALL CRT$PRINT$STRING(.MSG2);
```

```
361   2        CALL SEND$CRLF;
362   2        CALL GET$LONG(.SYSTEM.LONG);
363   2        CALL CRT$PRINT$STRING(.MSG0);
364   2        CALL SEND$CRLF;
365   2        CALL CRT$PRINT$STRING(.MSG3);
366   2        CALL SEND$CRLF;
367   2        OWN$SHIP(0).TIME(0) = HOURS;
368   2        OWN$SHIP(0).TIME(1) = MINUTES;
369   2        OWN$SHIP(0).TIME(2) = SECONDS;
370   2        CALL GET$LAT(.OWN$SHIP$INFO.LAT);
371   2        CALL CRT$PRINT$STRING(.MSG0);
372   2        CALL SEND$CRLF;
373   2        CALL CRT$PRINT$STRING(.MSG3);
374   2        CALL SEND$CRLF;
375   2        CALL GET$LONG(.OWN$SHIP$INFO.LONG);
376   2        CALL CRT$PRINT$STRING(.MSG0);
377   2        CALL SEND$CRLF;
378   2        OWN$SHIP$INFO.POINTER = 0;
379   2        CALL CONV$XY(.OWN$SHIP$INFO.LAT, .OWN$SHIP$INFO.LONG, .OWN
      -       $SHIP(0).X,
                          .OWN$SHIP(0).Y);
380   2        CALL CRT$PRINT$STRING(.MSG3);
381   2        CALL SEND$CRLF;
382   2        CALL GET$COURSE$BRG(0, .OWN$SHIP(0).CRS);
383   2        CALL CRT$PRINT$STRING(.MSG0);
384   2        CALL SEND$CRLF;
385   2        CALL CRT$PRINT$STRING(.MSG3);
386   2        CALL SEND$CRLF;
387   2        CALL GET$SPEED(.OWN$SHIP(0).SPD);
388   2        CALL CRT$PRINT$STRING(.MSG0);
```

```
389    2          CALL SEND$CRLF;
390    2          CALL CRT$PRINT$STRING(.MSG4);
391    2          CALL SEND$CRLF;
392    2          CALL GET$SCALE(.SYSTEM.SCALE);
393    2          CALL CRT$PRINT$STRING(.MSG0);
394    2          CALL SEND$CRLF;
395    2          CALL START$BLINK;
396    2          CALL CRT$PRINT$STRING(.MSG5);
397    2          CALL CRT$WRITE(18H);              /*   STOP BLINK MODE   */
398    2          CHAR = 0;
399    2          DO WHILE CHAR <> 2CH;             /*   WAIT FOR ORDER TO STA
            -   RT   */
400    3             IF CHAR <> 2CH THEN CALL SEND$BEL;
402    3             CHAR = CRT$READ;
403    3             END;
404    2          CALL INITIATE$CLOCK;
405    2          CALL CLEAR$LOW$SCREEN;
406    2          CALL ACTUAL$TIME;
407    2          CALL PRINT$TIME(.TIME$BUFFER);
408    2          CALL PRINT$TIME$ZONE(.SYSTEM.NUM$ZONE);
409    2          CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LAT, .LAT$STRING, 0);
410    2          CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LONG, .LONG$STRING, 1)
            -   ;
411    2          CALL PRINT$LAT$LONG(.LAT$STRING, .LONG$STRING);
412    2          CHAR = FP$FORMAT(.OWN$SHIP(0).CRS, .CRS$STRING, 3, 1);
413    2          CALL PRINT$COURSE(.CRS$STRING);
414    2          CHAR = FP$FORMAT(.OWN$SHIP(0).SPD, .SPD$STRING, 2, 1);
415    2          CALL PRINT$SPEED(.SPD$STRING);
416    2          DO CHAR = 0 TO 5;
417    3             CONTACTS$STRING(CHAR) = '0';
```

```
418   3            END;
419   2         CONTACTS$STRING(6), CONTACTS$STRING(7) = '$';
420   2         CALL PRINT$CONTACTS(.CONTACTS$STRING);
421   2         CALL PRINT$MODE(.INPUT$MODE);
422   2         CONTACT$INFO$STRING(44), CONTACT$INFO$STRING(45) = '$';
423   2         CALL CRT$WRITE(PROMPT);
424   2         END GET$SYSTEM$PARAMETERS;


                /*************************************************************
            -    ***************
                 *
                 * DISPLAY$CONTACT:
                 *   THIS PROCEDURE IS USED TO DISPLAY ALL THE AVAILABLE INFOR
            -    MATION ABOUT
                 *   A GIVEN CONTACT, ACCORDING TO THE FORMAT PRESENT AT THE C
            -    RT.
                 *
                 * PARAMETERS:
                 *    - ROW.- INDICATES IN WHICH DISPLAY ROW TO PUT THE INFORMA
            -    TION.
                 *    - INDEX.- SHOWS THE RELATIVE POSITION OF THE CONTACT IN T
            -    HE DATA STRUC-
                 *      TURES BEING USED.
                 *
                 **************************************************************
            -    **********/
```

```
425   1       DISPLAY$CONTACT: PROCEDURE (ROW, INDEX) PUBLIC;
426   2         DCL KIND (*) BYTE DATA ('FRIHOSUNK'),
                    (ROW,INDEX,I,J,TEMP,TEST) BYTE;
427   2         CONTACT$DISPLAY(ROW) = INDEX;
428   2         ROW = ROW + 1;
429   2         CALL DE$HASH(CONTACT$INFO(INDEX).DESIG,
                             .CONTACT$INFO$STRING(Ø));
430   2         CONTACT$INFO$STRING(2), CONTACT$INFO$STRING(3) = 'Q';
431   2           IF CONTACT$INFO(INDEX).TYPE = Ø
                   THEN DO;
433   3             CONTACT$INFO$STRING(4) = 'S';
434   3             CONTACT$INFO$STRING(5) = 'U';
435   3             END;
436   2            ELSE DO;
437   3             IF CONTACT$INFO(INDEX).TYPE = 1
                       THEN DO;
439   4               CONTACT$INFO$STRING(4), CONTACT$INFO$STRING(5) = '
       -  S';
440   4               END;
441   3              ELSE DO;
442   4               CONTACT$INFO$STRING(4), CONTACT$INFO$STRING(5) = '
       -  A';
443   4               END;
444   3               END;
445   2         J = CONTACT$INFO(INDEX).KIND;
446   2         DO I = Ø TO 2;
447   3            CONTACT$INFO$STRING(6 + I) = KIND((3 * J) + I);
448   3            END;
449   2         J = CONTACT$INFO(INDEX).POINTER;
45Ø   2         CONTACT$INFO$STRING(9) = CONTACT$POSI(J).TIME(Ø) / 1Ø + 3Ø
```

```
               -   H;
 451    2            CONTACT$INFO$STRING(10) = CONTACT$POSI(J).TIME(0) MOD 10 +
               -    30H;
 452    2            CONTACT$INFO$STRING(11) = CONTACT$POSI(J).TIME(1) / 10 + 3
               -   0H;
 453    2            CONTACT$INFO$STRING(12) = CONTACT$POSI(J).TIME(1) MOD 10 +
               -    30H;
 454    2            TEMP = FP$FORMAT(.CONTACT$POSI(J).BRG, .CONTACT$INFO$STRIN
               -   G(13), 3, 1);
 455    2            CALL RANGE$FORMAT(.CONTACT$POSI(J).RNG, .CONTACT$INFO$STRI
               -   NG(17));
 456    2            TEMP = GET$CPA(INDEX, .CONTACT$INFO$STRING(23));
 457    2            IF TEMP = 0
                     THEN DO;
 459    3               IF CONTACT$INFO(INDEX).CRS$FLAG
                        THEN DO;
 461    4                  TEST = FP$FORMAT(.CONTACT$POSI(J).CRS,
                                              .CONTACT$INFO$STRING(23), 3, 1);
 462    4                 END;
 463    3               ELSE DO;
 464    4                 DO I = 23 TO 26;
 465    5                    CONTACT$INFO$STRING(I) = ' ';
 466    5                    END;
 467    4                 END;
 468    3               IF CONTACT$INFO(INDEX).SPD$FLAG
                        THEN DO;
 470    4                  TEST = FP$FORMAT(.CONTACT$POSI(J).SPD,
                                              .CONTACT$INFO$STRING(27), 2, 1);
 471    4                 END;
 472    3               ELSE DO;
 473   4               DO I = 27 TO 29;
 474   5                  CONTACT$INFO$STRING(I) = ' ';
 475   5                  END;
 476   4               END;
 477   3            DO I = 30 TO 43;
 478   4               CONTACT$INFO$STRING(I) = ' ';
 479   4               END;
 480   3            END;
 481   2         CONTACT$INFO$STRING(44), CONTACT$INFO$STRING(45) = '$';
 482   2         CALL PRINT$CONTACT$INFO(ROW, .CONTACT$INFO$STRING);
 483   2         END DISPLAY$CONTACT;


                      /***********************************************************
                -      ***************
                      *
                      * CREATE:
                      *   THIS PROCEDURE IS USED TO OBTAIN ALL PERTINENT INFORMATIO
                -      N ABOUT A NEW
                      *   CONTACT.
                      *
                      * USAGE:
                      *   UNTYPED PROCEDURE. THIS PROCEDURE SHOULD BE CALLED AFTER
                -      IT HAS BEEN
                      *   DETERMINED THAT A CONTACT CAN BE ACCEPTED IN THE SYSTEM,
                -      AND THAT THE
                      *   NUMBER OF CONTACTS HAS BEEN UPDATED.
                      *
```

```
                    *****************************************************************
                -    **************/
 484   1            CREATE: PROCEDURE PUBLIC;
 485   2              DCL STR1 (*) BYTE DATA ('ARE THE FOLLOWING VALUES KNOWN?$$
                -    '),
                          ARRAY (2) BYTE,
                          A ADDRESS,
                          (OK, TEMP, I, J, INDEX, H, M, S) BYTE;
 486   2              H = HOURS;     /* TO SAVE THE TIME OF CALL */
 487   2              M = MINUTES;
 488   2              S = SECONDS;
                         /* UPDATE POSITION OF OWN SHIP */
 489   2              CALL MOVE$OWN$SHIP;
                         /* GET INITIAL CONTACT VALUES */
 490   2              OK = 0;
 491   2              DO WHILE OK = 0;
 492   3                 CALL CRT$PRINT$STRING(.TITLE$0);
 493   3                 CALL SEND$CRLF;
 494   3                 CALL CRT$PRINT$STRING(.STR1);
 495   3                 CALL SEND$CRLF;
 496   3                 CALL CRT$PRINT$STRING(.MSG$2);
 497   3                 ARRAY(0) = CHECK$YES$NO;
 498   3                 CALL SEND$CRLF;
 499   3                 CALL CRT$PRINT$STRING(.MSG$3);
 500   3                 ARRAY(1) = CHECK$YES$NO;
 501   3                 CALL SEND$CRLF;
 502   3                 OK = CHECK$INPUT;
 503   3                 CALL CLEAR$LOW$SCREEN;
 504   3                 END;
 505   2              TEMP, I = 0;
```

AD-A102 693 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
MICROCOMPUTER BASED SHIP-BOARD GUN CONTROL SYSTEM.(U)
MAR 81 A ERDOGAN

UNCLASSIFIED NL

6 OF 7

AD
A102693

```
 506   2        DO WHILE TEMP = 0;
 507   3           IF CONTACT$INFO(I).DESIG = 0
                   THEN DO;
 509   4              TEMP = 1;
 510   4              INDEX = I;
 511   4              END;
 512   3           I = I + 1;
 513   3           END;
 514   2        OK = 0;
 515   2        DO WHILE OK = 0;      /* GET DESIG */
 516   3           CALL CRT$PRINT$STRING(.TITLE$0);
 517   3           CALL SEND$CRLF;
 518   3           A, CONTACT$INFO(INDEX).DESIG = GET$DESIG;
 519   3           DO I = 0 TO 14;
 520   4              IF (I <> INDEX) AND (A = CONTACT$INFO(I).DESIG)
                      THEN DO;
 522   5                 CALL CRT$PRINT$STRING(.('DESIGNATION ALREADY IN U
     -  SE.$$'));
 523   5                 CALL SEND$CRLF;
 524   5                 CALL CHECK$GO$KEY;
 525   5                 OK = 0;
 526   5                 CALL CLEAR$LOW$SCREEN;
 527   5                 GO TO L;
 528   5                 END;
 529   4              OK = 1;
 530   4              END;
 531   3        L: END;
 532   2        CALL CRT$PRINT$STRING(.TITLE$0);
 533   2        CALL SEND$CRLF;
 534   2        CONTACT$INFO(INDEX).TYPE = GET$TYPE;   /* GET TYPE */
 535   2          CALL CRT$PRINT$STRING(.TITLE$0);
 536   2          CALL SEND$CRLF;
 537   2          TEMP, CONTACT$INFO(INDEX).KIND = GET$KIND;   /* GET KIND *
             -   /
 538   2          SYSTEM.CONTACT$KIND(TEMP) = SYSTEM.CONTACT$KIND(TEMP) + 1;
 539   2          CALL CRT$PRINT$STRING(.TITLE$0);
 540   2          CALL SEND$CRLF;
 541   2          J, CONTACT$INFO(INDEX).POINTER = 15*INDEX;
 542   2          CONTACT$INFO(INDEX).OS$POINTER = 0FFH;
 543   2          CONTACT$INFO(INDEX).FLAG = 0;
 544   2          CONTACT$INFO(INDEX).CRS$FLAG = 0;
 545   2          CONTACT$INFO(INDEX).SPD$FLAG = 0;
 546   2          CONTACT$POSI(J).TIME(0) = H;
 547   2          CONTACT$POSI(J).TIME(1) = M;
 548   2          CONTACT$POSI(J).TIME(2) = S;
 549   2          CALL GET$COURSE$BRG(1, .CONTACT$POSI(J).BRG);
 550   2          CALL CRT$PRINT$STRING(.TITLE$0);
 551   2          CALL SEND$CRLF;
 552   2          CALL GET$RANGE(.CONTACT$POSI(J).RNG);
 553   2          CALL CONV$REL$XY(.CONTACT$POSI(J).BRG, .CONTACT$POSI(J).RN
             -   G,
                                  .CONTACT$POSI(J).X, .CONTACT$POSI(J).Y);
 554   2          IF ARRAY(0)
                  THEN DO;
 556   3             CONTACT$INFO(INDEX).CRS$FLAG = 1;
 557   3             CALL CRT$PRINT$STRING(.TITLE$0);
 558   3             CALL SEND$CRLF;
 559   3             CALL GET$COURSE$BRG(0,.CONTACT$POSI(J).CRS);
 560   3             END;
 561   2          ELSE CONTACT$INFO(INDEX).CRS$FLAG = 0;
 562   2          IF ARRAY(1)
                  THEN DO;
 564   3             CONTACT$INFO(INDEX).SPD$FLAG = 1;
 565   3             CALL CRT$PRINT$STRING(.TITLE$0);
 566   3             CALL SEND$CRLF;
 567   3             CALL GET$SPEED(.CONTACT$POSI(J).SPD);
 568   3             END;
 569   2          ELSE CONTACT$INFO(INDEX).SPD$FLAG = 0;
 570   2          I, OK = 0;
 571   2          DO WHILE (OK = 0) AND ( I <= LAST(CONTACT$DISPLAY));
 572   3             IF CONTACT$DISPLAY(I) = 0FFH
                     THEN DO;
 574   4                OK = 1;
 575   4                CALL DISPLAY$CONTACT(I, INDEX);
 576   4                END;
 577   3             I = I + 1;
 578   3             END;
 579   2          CALL DISPLAY$KIND;
 580   2          CALL PLASMA$CONTACT(INDEX);
 581   2          END CREATE;


                 /**************************************************************
              -   **************
                  *
                  * REMOVE:
                  *    THIS PROCEDURE IS USED TO REMOVE A CONTACT FROM THE SYSTE
              -   M.
```

```
                   *
                   ******************************************************************
              -     ***************/
 582    1          REMOVE: PROCEDURE PUBLIC;
 583    2             DCL DESIG ADDRESS,
                          STRING(4) BYTE,
                          (ROW, OK, I, CHAR, TEMP, CLASS, CHECK) BYTE;
 584    2             OK = 0;
 585    2             DO WHILE OK = 0;
 586    3                CALL CRT$PRINT$STRING(.TITLE$1);
 587    3                CALL SEND$CRLF;
 588    3                CALL CRT$PRINT$STRING(.
                             ('ARE YOU SURE YOU WANT TO DELETE A CONTACT? (Y/N)
              -     $$'));
 589    3                IF ((CHAR:= CHECK$YES$NO) = 0 )      /* NOT SURE */
                         THEN DO;
 591    4                    CALL CLEAR$LOW$SCREEN;
 592    4                    RETURN;
 593    4                    END;
 594    3                CALL SEND$CRLF;
 595    3                DESIG = GET$DESIG;
 596    3                CALL DE$HASH(DESIG, .STRING);
 597    3                STRING(2), STRING(3) = '$';
 598    3                CALL CRT$PRINT$STRING(.TITLE$1);
 599    3                CALL SEND$CRLF;
 600    3                IF (TEMP:= CHECK$DESIG(DESIG)) <> 0FFH
                         THEN DO;
 602    4                    CALL CRT$PRINT$STRING(.('CONTACT TO BE DELETED: $$
              -     '));
 603    4                    CALL CRT$PRINT$STRING(.STRING);
```

```
604    4                    OK = 1;
605    4                    END;
606    3               ELSE DO;
607    4                    CALL CRT$PRINT$STRING(.('DESIG NOT IN USE. $$'));
608    4                    END;
609    3               CALL SEND$CRLF;
610    3               CALL SEND$CRLF;
611    3               CALL CHECK$GO$KEY;
612    3               CALL CLEAR$LOW$SCREEN;
613    3               END;
614    2            CONTACT$INFO(TEMP).DESIG = 00H;
615    2            SYSTEM.NUMCTS = SYSTEM.NUMCTS - 1;
616    2            CLASS = CONTACT$INFO(TEMP).KIND;
617    2            SYSTEM.CONTACT$KIND(CLASS) = SYSTEM.CONTACT$KIND(CLASS) -
            -    1;
618    2            ROW = 0FFH;
619    2            DO I = 0 TO 5;
620    3               IF CONTACT$DISPLAY(I) = TEMP
                       THEN DO;
622    4                    CONTACT$DISPLAY(I) = 0FFH;
623    4                    ROW = I;
624    4                    END;
625    3               END;
626    2            CALL PLASMA$DELETE(TEMP);
627    2            IF (SYSTEM.NUMCTS > 5) AND (ROW <> 0FFH)
                    THEN DO;
629    3                  OK = 0;
630    3                  DO WHILE OK = 0;
631    4                    CALL CRT$PRINT$STRING(.
                                ('ENTER THE DESIG OF A CONTACT DESIRED TO BE D
              -   ISPLAYED: $$'));
632    4                    CALL SEND$CRLF;
633    4                    DESIG = GET$DESIG;
634    4                    CALL DE$HASH(DESIG, .STRING);
635    4                    STRING(2), STRING(3) = '$';
636    4                    TEMP = CHECK$DESIG(DESIG);
637    4                    IF TEMP = 0FFH
                            THEN DO;
639    5                       CALL CRT$PRINT$STRING(.('DESIG $$'));
640    5                       CALL CRT$PRINT$STRING(.STRING);
641    5                       CALL CRT$PRINT$STRING(.(' NOT IN USE.$$'));
642    5                       CALL SEND$CRLF;
643    5                       END;
644    4                    ELSE DO;
645    5                       CHECK = 0;
646    5                       DO I = 0 TO LAST(CONTACT$DISPLAY);
647    6                         IF CONTACT$DISPLAY(I) = TEMP
                                 THEN DO;
649    7                           CALL CRT$PRINT$STRING(.('CONTACT ALREAD
              -   Y DISPLAYED.$$'));
650    7                           CALL SEND$CRLF;
651    7                           CHECK = 1;
652    7                           I = LAST(CONTACT$DISPLAY) + 2;
653    7                           END;
654    6                         END;
655    5                       IF CHECK = 0
                               THEN DO;
657    6                         CALL CRT$PRINT$STRING(.('CONTACT $$'));
658    6                         CALL CRT$PRINT$STRING(.STRING);
659    6                         CALL CRT$PRINT$STRING(.(' WILL BE DISPLAYE
             -   D.$$'));
 66Ø    6                             CALL SEND$CRLF;
 661    6                             OK = 1;
 662    6                             END;
 663    5                          END;
 664    4                     CALL SEND$CRLF;
 665    4                     CALL CHECK$GO$KEY;
 666    4                     CALL CLEAR$LOW$SCREEN;
 667    4                     END;
 668    3                  IF TEMP <> ØFFH
                           THEN DO;
 67Ø    4                       OK = 1;
 671    4                       CALL DISPLAY$CONTACT(ROW, TEMP);
 672    4                       END;
 673    3                  END;
 674    2             ELSE DO;
 675    3                  IF ROW <> ØFFH
                           THEN DO;
 677    4                       DO I = Ø TO 41;
 678    5                          CONTACT$INFO$STRING(I) = ' ';
 679    5                          END;
 68Ø    4                       CONTACT$INFO$STRING(42), CONTACT$INFO$STRING(43)
             -    = '$';
 681    4                       CALL PRINT$CONTACT$INFO(ROW + 1, .CONTACT$INFO$S
             -   TRING);
 682    4                       END;
 683    3                  END;
 684    2             CALL DISPLAY$KIND;
 685    2             END REMOVE;
```

```
                 /**************************************************************
            -     ***************
                 *
                 * REDESIGNATE:
                 *   THIS PROCEDURE IS USED TO CHANGE THE DESIG OF A CONTACT.
                 *
                 ***************************************************************
            -     *****************/
 686   1         REDESIGNATE: PROCEDURE PUBLIC;
 687   2            DCL (NEW,OLD) ADDRESS,
                        STR1 (4) BYTE,
                        STR2 (4) BYTE,
                        (TEMP, TEMP1, INDEX, I) BYTE;
 688   2            TEMP = 0FFH;
 689   2            DO WHILE TEMP = 0FFH;
 690   3               CALL CRT$PRINT$STRING(.TITLE$2);
 691   3               CALL SEND$CRLF;
 692   3               CALL CRT$PRINT$STRING(.('ENTER OLD DESIG AS REQUESTED:$
            -     $'));
 693   3               CALL SEND$CRLF;
 694   3               OLD = GET$DESIG;
 695   3               TEMP = CHECK$DESIG(OLD);
 696   3               IF TEMP = 0FFH
                       THEN DO;
 698   4                  CALL CRT$PRINT$STRING(.('DESIG NOT IN USE.$$'));
 699   4                  CALL SEND$CRLF;
 700   4                  CALL CHECK$GO$KEY;
 701   4                  CALL CLEAR$LOW$SCREEN;
```

```
702   4              END;
703   3            END;
704   2          TEMP1 = 0;
705   2          DO WHILE TEMP1 <> 0FFH;
706   3            CALL CRT$PRINT$STRING(.TITLE$2);
707   3            CALL SEND$CRLF;
708   3            CALL CRT$PRINT$STRING(.('ENTER NEW DESIG AS REQUESTED:$
      -    $'));
709   3            CALL SEND$CRLF;
710   3            NEW = GET$DESIG;
711   3            TEMP1 = CHECK$DESIG(NEW);
712   3            IF TEMP1 <> 0FFH
                   THEN DO;
714   4              CALL CRT$PRINT$STRING(.('DESIG ALREADY IN USE.$$'));
715   4              CALL SEND$CRLF;
716   4              END;
717   3            ELSE DO;
718   4              STR1(2), STR1(3), STR2(2), STR2(3) = '$';
719   4              CALL DE$HASH(OLD, .STR1);
720   4              CALL DE$HASH(NEW, .STR2);
721   4              CALL CRT$PRINT$STRING(.('CONTACT $$'));
722   4              CALL CRT$PRINT$STRING(.STR1);
723   4              CALL CRT$PRINT$STRING(.(' WILL BE CHANGED TO $$'));
724   4              CALL CRT$PRINT$STRING(.STR2);
725   4              CALL CRT$WRITE('.');
726   4              CALL SEND$CRLF;
727   4              CONTACT$INFO(TEMP1).DESIG = NEW;
728   4              CALL SEND$CRLF;
729   4              END;
730   3            CALL CHECK$GO$KEY;
```

```
 731   3             CALL CLEAR$LOW$SCREEN;
 732   3             IF TEMP1 = 0FFH
                     THEN DO;
 734   4                 DO I = 0 TO LAST(CONTACT$DISPLAY);
 735   5                     IF CONTACT$DISPLAY(I) = TEMP
                             THEN DO;
 737   6                         INDEX = CONTACT$DISPLAY(I);
 738   6                         CONTACT$INFO(INDEX).DESIG = NEW;
 739   6                         CALL PRINT$CONTACT$INFO(I + 1, .STR2);
 740   6                         END;
 741   5                     END;
 742   4                 END;
 743   3             END;
 744   2         CALL PLASMA$REDESIG(TEMP1);
 745   2         END REDESIGNATE;


                 /*************************************************************
             -    ****************
                 *
                 * UPDATE:
                 *   THIS PROCEDURE IS USED TO UPDATE INFORMATION ABOUT ANY CO
             -    NTACT.
                 *
                 **************************************************************
             -    ****************/
 746   1         UPDATE: PROCEDURE PUBLIC;
 747   2             DCL DESIG ADDRESS,
                         ARRAY (6) BYTE,
                        (LAST$INFO,OK,I,J,K,TEMP,KIND,OLD$KIND,TYPE,INDEX,COUN
              -     T,H,M,S) BYTE;
 748    2             H = HOURS;        /* SAVE TIME OF CALL */
 749    2             M = MINUTES;
 75Ø    2             S = SECONDS;
                        /* UPDATE OWN SHIP POSITION */
 751    2             CALL MOVE$OWN$SHIP;
                        /* GET CONTACT VALUES */
 752    2             OK = ØFFH;
 753    2             DO WHILE OK = ØFFH;
 754    3                CALL CRT$PRINT$STRING(.TITLE$3);
 755    3                CALL SEND$CRLF;
 756    3                CALL CRT$PRINT$STRING(.('ENTER CONTACT DESIG AS REQUEST
              -     ED:$$'));
 757    3                CALL SEND$CRLF;
 758    3                DESIG = GET$DESIG;
 759    3                INDEX = CHECK$DESIG(DESIG);
 76Ø    3                IF INDEX = ØFFH
                         THEN DO;
 762    4                   CALL CRT$PRINT$STRING(.('DESIG NOT IN USE.$$'));
 763    4                   CALL SEND$CRLF;
 764    4                   CALL CHECK$GO$KEY;
 765    4                   CALL CLEAR$LOW$SCREEN;
 766    4                   END;
 767    3                OK = INDEX;
 768    3                END;
 769    2             OK = Ø;
 77Ø    2             DO WHILE OK = Ø;
 771    3                CALL CRT$PRINT$STRING(.TITLE$3);
 772    3                CALL SEND$CRLF;
```

```
            CALL CRT$PRINT$STRING(.MSG$1);
            CALL SEND$CRLF;
            CALL CRT$PRINT$STRING(.MSG$9);
            ARRAY (0) = CHECK$YES$NO;
            CALL CRT$PRINT$STRING(.BLANK);
            CALL CRT$PRINT$STRING(.MSG$A);
            ARRAY (1) = CHECK$YES$NO;
            CALL SEND$CRLF;
            CALL CRT$PRINT$STRING(.MSG$6);
            ARRAY (2) = CHECK$YES$NO;
            CALL CRT$PRINT$STRING(.BLANK);
            CALL CRT$PRINT$STRING(.MSG$7);
            ARRAY (3) = CHECK$YES$NO;
            CALL SEND$CRLF;
            CALL CRT$PRINT$STRING(.MSG$2);
            ARRAY (4) = CHECK$YES$NO;
            CALL CRT$PRINT$STRING(.BLANK);
            CALL CRT$PRINT$STRING(.MSG$3);
            ARRAY (5) = CHECK$YES$NO;
            CALL SEND$CRLF;
            OK = CHECK$INPUT;
            CALL CLEAR$LOW$SCREEN;
            END;
         TEMP = 0;
         DO I = 0 TO LAST(ARRAY);
            IF ARRAY(I) THEN TEMP = 1;
            END;
         IF TEMP = 0 THEN RETURN;   /* NO INPUT IS DESIRED */
         J, LAST$INFO = CONTACT$INFO(INDEX).POINTER;
         IF ARRAY(2) OR ARRAY(3)
```

```
                       THEN DO;
 8Ø6    3                 CONTACT$INFO(INDEX).POINTER = CONTACT$INFO(INDEX).POINT
                -     ER + 1;
 8Ø7    3                 IF CONTACT$INFO(INDEX).POINTER = 15*INDEX + 15
                          THEN DO;
 8Ø9    4                    CONTACT$INFO(INDEX).POINTER = 15*INDEX;
 81Ø    4                    CONTACT$INFO(INDEX).FLAG = ØFFH;
 811    4                    END;
 812    3                 IF CONTACT$INFO(INDEX).OS$POINTER <> ØFFH
                          THEN DO;
 814    4                       IF CONTACT$INFO(INDEX).FLAG
                                THEN DO;
 816    5                            TEMP = CONTACT$INFO(INDEX).OS$POINTER;
 817    5                            COUNT = 1;
 819    5                            DO WHILE TEMP <> CONTACT$INFO(INDEX).POINTER;
 819    6                               IF TEMP = (INDEX + 1)*15
                                        THEN TEMP = INDEX*15;
 821    6                               ELSE TEMP = TEMP + 1;
 822    6                               COUNT = COUNT + 1;
 823    6                               END;
 824    5                            END;
 825    4                       ELSE DO;
 826    5                            COUNT = CONTACT$INFO(INDEX).POINTER -
                                             CONTACT$INFO(INDEX).OS$POINTER;
 827    5                            END;
 828    4                       IF COUNT >= 5
                                THEN CONTACT$INFO(INDEX).OS$POINTER = ØFFH;
 83Ø    4                    END;
 831    3                 J = CONTACT$INFO(INDEX).POINTER;
 832    3                 CONTACT$POSI(J).TIME(Ø) = H;
 833   3           CONTACT$POSI(J).TIME(1) = M;
 834   3           CONTACT$POSI(J).TIME(2) = S;
 835   3           END;
 836   2         DO I = 0 TO LAST(ARRAY);
 837   3           IF ARRAY(I)
                   THEN DO;
 839   4             CALL CRT$PRINT$STRING(.TITLE$3);
 840   4             CALL SEND$CRLF;
 841   4             DO CASE I;
 842   5               DO;
 843   6                 CONTACT$INFO(INDEX).TYPE = GET$TYPE;
 844   6               END;
 845   5               DO;
 846   6                 OLD$KIND = CONTACT$INFO(INDEX).KIND;
 847   6                 KIND, CONTACT$INFO(INDEX).KIND = GET$KIND;
 848   6                 SYSTEM.CONTACT$KIND(OLD$KIND) = SYSTEM.CONTACT
      -   $KIND(OLD$KIND) -1;
 849   6                 SYSTEM.CONTACT$KIND(KIND) = SYSTEM.CONTACT$KIN
      -   D(KIND) + 1;
 850   6               END;
 851   5               DO;
 852   6                 CALL GET$COURSE$BRG(1, .CONTACT$POSI(J).BRG);
 853   6               END;
 854   5               DO;
 855   6                 CALL GET$RANGE(.CONTACT$POSI(J).RNG);
 856   6               END;
 857   5               DO;
 858   6                 CALL GET$COURSE$BRG(0, .CONTACT$POSI(J).CRS);
 859   6                 CONTACT$INFO(INDEX).CRS$FLAG = 1;
 860   6               END;
```

```
861   5                 DO;
862   6                    CALL GET$SPEED(.CONTACT$POSI(J).SPD);
863   6                    CONTACT$INFO(INDEX).SPD$FLAG = 1;
864   6                 END;
865   5                 END;    /* END CASE */
866   4              END;       /* IF THEN */
867   3           ELSE DO;
868   4              IF ARRAY(2) OR ARRAY(3)
                     THEN DO;
870   5              DO CASE I;
871   6                 DO;
872   7                 END;
873   6                 DO;
874   7                 END;
875   6                 DO;
876   7                    IF (NOT ARRAY(2)) AND ARRAY(3)
                           THEN DO;
878   8                       DO K = Ø TO 3;
879   9                          CONTACT$POSI(J).BRG(K) = CONTACT$POSI(LA
    -    ST$INFO).BRG(K);
880   9                          END;
881   8                       END;
882   7                 END;
883   6                 DO;
884   7                    IF (NOT ARRAY(3)) AND ARRAY(2)
                           THEN DO;
886   8                       DO K = Ø TO 3;
887   9                          CONTACT$POSI(J).RNG(K) = CONTACT$POSI(LA
    -    ST$INFO).RNG(K);
888   9                          END;
```

```
889    8                        END;
890    7                    END;
891    6                    DO;
892    7                      DO K = 0 TO 3;
893    8                        CONTACT$POSI(J).CRS(K) = CONTACT$POSI(LAST$
         -    INFO).CRS(K);
894    8                        END;
895    7                    END;
896    6                    DO;
897    7                      DO K = 0 TO 3;
898    8                        CONTACT$POSI(J).SPD(K) = CONTACT$POSI(LAST$
         -    INFO).SPD(K);
899    8                        END;
900    7                    END;
901    5                    END;   /* END CASE */
902    5                    END;   /* IF THEN */
903    4                  END;     /* ELSE */
904    3                END;       /* END DO */
905    2              IF ARRAY(2) OR ARRAY(3)
                        THEN DO;
907    3                   CALL CONV$REL$XY(.CONTACT$POSI(J).BRG, .CONTACT$PO
         -    SI(J).RNG,
                                          .CONTACT$POSI(J).X,   .CONTACT$PO
         -    SI(J).Y);
908    3                   CALL PLASMA$CONTACT(INDEX);
909    3                   END;
910    2              IF ARRAY (1)
                        THEN CALL DISPLAY$KIND;
912    2              I, OK = 0;
913    2              DO WHILE (OK = 0) AND (I <= LAST(CONTACT$DISPLAY));
```

```
 914   3            IF CONTACT$DISPLAY(I) = INDEX
                    THEN DO;
 916   4               OK = 1;
 917   4               CALL DISPLAY$CONTACT(I, INDEX);
 918   4               END;
 919   3            I = I + 1;
 920   3            END;
 921   2         END UPDATE;


                /************************************************************
              -  ***************
                *
                * SWAP$CONTACTS:
                *   THIS PROCEDURE IS USED TO SWAP ONE CONTACT BEING DISPLAYE
              -  D WITH
                *   ANOTHER WICH IS IN THE SYSTEM BUT NOT AT THE DISPLAY.
                *
                *************************************************************
              -  ***************/
 922   1        SWAP$CONTACTS: PROCEDURE PUBLIC;
 923   2           DCL (CONTACT$IN, CONTACT$OUT) ADDRESS,
                        STRING (4) BYTE,
                        (TEMP, TEMP1, INDEX, I, J) BYTE;
 924   2           TEMP, TEMP1, J = 0FFH;
 925   2           DO WHILE TEMP = 0FFH;
 926   3              CALL CRT$PRINT$STRING(.TITLE$5);
 927   3              CALL SEND$CRLF;
 928   3              CALL CRT$PRINT$STRING(.('ENTER CONTACT TO BE OUT OF DIS
             -   PLAY:$$'));
 929   3               CALL SEND$CRLF;
 930   3               CONTACT$OUT = GET$DESIG;
 931   3               TEMP = CHECK$DESIG(CONTACT$OUT);
 932   3               IF TEMP = 0FFH
                       THEN DO;
 934   4                    CALL CRT$PRINT$STRING(.('CONTACT NOT IN SYSTEM.$$'
             -   ));
 935   4                    CALL SEND$CRLF;
 936   4                    CALL CHECK$GO$KEY;
 937   4                    CALL CLEAR$LOW$SCREEN;
 938   4                    END;
 939   3               ELSE DO;
 940   4                    DO I = 0 TO LAST(CONTACT$DISPLAY);
 941   5                       IF CONTACT$DISPLAY(I) = TEMP
                               THEN DO;
 943   6                            J = I;
 944   6                            I = 6;
 945   6                            END;
 946   5                       END;
 947   4                    IF J = 0FFH
                            THEN DO;
 949   5                         TEMP = 0FFH;
 950   5                         CALL CRT$PRINT$STRING(.('CONTACT NOT AT DISPL
             -   AY.$$'));
 951   5                         CALL SEND$CRLF;
 952   5                         CALL CHECK$GO$KEY;
 953   5                         CALL CLEAR$LOW$SCREEN;
 954   5                         END;
 955   4                    END;
```

```
956   3             END;
957   2          DO WHILE TEMP1 = 0FFH;
958   3             CALL CRT$PRINT$STRING(.TITLE$5);
959   3             CALL SEND$CRLF;
960   3             CALL CRT$PRINT$STRING(.('ENTER CONTACT TO BE IN THE DIS
            -   PLAY:$$'));
961   3             CALL SEND$CRLF;
962   3             CONTACT$IN = GET$DESIG;
963   3             TEMP1 = CHECK$DESIG (CONTACT$IN);
964   3             IF TEMP1 = 0FFH
                    THEN DO;
966   4                   CALL CRT$PRINT$STRING(.('CONTACT NOT IN SYSTEM.$$'
            -   ));
967   4                   CALL SEND$CRLF;
968   4                   CALL CHECK$GO$KEY;
969   4                   CALL CLEAR$LOW$SCREEN;
970   4                   END;
971   3             ELSE DO;
972   4                   DO I = 0 TO LAST (CONTACT$DIS   Y);
973   5                      IF CONTACT$DISPLAY(I) = TE
                             THEN DO;
975   6                            TEMP1 = 0FFH;
976   6                            CALL CRT$PRINT$STRING(.('CONTACT ALREADY D
            -   ISPLAYED.$$'));
977   6                            CALL SEND$CRLF;
978   6                            CALL CHECK$GO$KEY;
979   6                            CALL CLEAR$LOW$SCREEN;
980   6                            I = LAST(CONTACT$DISPLAY) + 2;
981   6                            END;
982   5                      END;
```

```
983   4                 END;
984   3              END;
985   2           CALL DISPLAY$CONTACT (J, TEMP1);
986   2           CALL CLEAR$LOW$SCREEN;
987   2           END SWAP$CONTACTS;


                 /*****************************************************************
             -    *****************
                 *
                 * TRANSLATE:
                 *   THIS PROCEDURE IS USED TO TRANSLATE ALL X/Y VALUES IN THE
             -    SYSTEM, BY GI-
                 *   VEN VALUES.
                 *
                 * PARAMETERS:
                 *    - A.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE CHANGE I
             -    N X IS LOCATED.
                 *    - B.- POINTER TO A FOUR BYTE VECTOR IN WHICH THE CHANGE I
             -    N Y IS LOCATED.
                 *    - TEMP.- CAN HAVE TWO VALUES:
                 *                - 0: DO NOT CHANGE LAST POSITION OF OWN SHIP.
                 *                - 1: CHANGE ALL X/Y VALUES WITHOUT EXCEPTION.
                 *
                 ******************************************************************
             -    *****************/
988   1          TRANSLATE: PROCEDURE (A, B, TEMP);
989   2             DCL (A, B) ADDRESS,
                          X$DELTA BASED A (4) BYTE,
                          Y$DELTA BASED B (4) BYTE,
                          (I, J, P, NUM$PTS, TEMP, FLAG) BYTE;
 990   2               IF OWN$SHIP$INFO.FLAG THEN NUM$PTS = 29;
 992   2                                       ELSE NUM$PTS = OWN$SHIP$INFO.POINTER
            -       ;
 993   2               DO I = 0 TO NUM$PTS;
 994   3                 FLAG = 0;
 995   3                 DO WHILE ((TEMP = 1) OR ((TEMP = 0) AND (I <> OWN$SHIP$
            -       INFO.POINTER)))
                                      AND (FLAG = 0);
 996   4                   CALL FADD(.OWN$SHIP(I).X, .X$DELTA, .OWN$SHIP(I).X);
 997   4                   CALL FADD(.OWN$SHIP(I).Y, .Y$DELTA, .OWN$SHIP(I).Y);
 998   4                   FLAG = 1;
 999   4                   END;
1000   3                 END;
1001   2               DO I = 0 TO 14;
1002   3                 IF CONTACT$INFO(I).DESIG <> 0
                         THEN DO;
1004   4                     IF CONTACT$INFO(I).FLAG THEN NUM$PTS = 14;
1006   4                                             ELSE NUM$PTS = CONTACT$INFO(I).
            -       POINTER MOD 15;
1007   4                     DO J = 0 TO NUM$PTS;
1008   5                       P = I*15 + J;
1009   5                       CALL FADD(.CONTACT$POSI(P).X, .X$DELTA, .CONTAC
            -       T$POSI(P).X);
1010   5                       CALL FADD(.CONTACT$POSI(P).Y, .Y$DELTA, .CONTAC
            -       T$POSI(P).Y);
1011   5                       END;
1012   4                     END;
1013   3                 END;
```

```
1014   2          END TRANSLATE;


                /**************************************************************
          -      ***************
                *
                * OWN$SHIP$UPDATE:
                *    THIS PROCEDURE IS USED TO UPDATE THE INFORMATION ABOUT TH
          -      E OWN SHIP.
                *
                ***************************************************************
          -      **************/
1015   1        OWN$SHIP$UPDATE: PROCEDURE PUBLIC;
1016   2           DCL ARRAY(4) BYTE,
                       OLD$LAT (4) BYTE,
                       OLD$LONG (4) BYTE,
                       X$DELTA (4) BYTE,
                       Y$DELTA (4) BYTE,
                       (LAST$INFO, OK, I, J, K, TEMP, H, M, S) BYTE;
1017   2           H = HOURS;    /* SAVE TIME OF CALL */
1018   2           M = MINUTES;
1019   2           S = SECONDS;
                       /* UPDATE OWN SHIP POSITION */
1020   2           CALL MOVE$OWN$SHIP;
                       /* GET OWN SHIP VALUES */
1021   2           OK = 0;
1022   2           DO WHILE OK = 0;
1023   3              CALL CRT$PRINT$STRING(.TITLE$4);
1024   3              CALL SEND$CRLF;
```

```
1025   3             CALL CRT$PRINT$STRING(.MSG$1);
1026   3             CALL SEND$CRLF;
1027   3             CALL CRT$PRINT$STRING(.MSG$4);
1028   3             ARRAY(0) = CHECK$YES$NO;
1029   3             CALL CRT$PRINT$STRING(.BLANK);
1030   3             CALL CRT$PRINT$STRING(.MSG$5);
1031   3             ARRAY(1) = CHECK$YES$NO;
1032   3             CALL SEND$CRLF;
1033   3             CALL CRT$PRINT$STRING(.MSG$2);
1034   3             ARRAY(2) = CHECK$YES$NO;
1035   3             CALL CRT$PRINT$STRING(.BLANK);
1036   3             CALL CRT$PRINT$STRING(.MSG$3);
1037   3             ARRAY(3) = CHECK$YES$NO;
1038   3             CALL SEND$CRLF;
1039   3             OK = CHECK$INPUT;
1040   3             CALL CLEAR$LOW$SCREEN;
1041   3             END;
1042   2          TEMP = 0;
1043   2          DO I = 0 TO LAST(ARRAY);
1044   3             IF ARRAY(I) THEN TEMP = 1;
1046   3             END;
1047   2          IF TEMP = 0 THEN RETURN;    /* NO INPUT IS DESIRED. */
                     /*   UPDATE OWN SHIP LAST POSITION    */
1049   2          CALL MOVE$OWN$SHIP;
1050   2          LAST$INFO = OWN$SHIP$INFO.POINTER;
1051   2          OWN$SHIP$INFO.POINTER = OWN$SHIP$INFO.POINTER + 1;
1052   2          IF OWN$SHIP$INFO.POINTER = 30
                  THEN DO;
1054   3             OWN$SHIP$INFO.POINTER = 0;
1055   3             OWN$SHIP$INFO.FLAG = 0FFH;
```

```
1056   3            END;
1057   2        J = OWN$SHIP$INFO.POINTER;
1058   2        OWN$SHIP(J).TIME(0) = H;
1059   2        OWN$SHIP(J).TIME(1) = M;
1060   2        OWN$SHIP(J).TIME(2) = S;
1061   2        IF ARRAY(0) OR ARRAY(1)
                THEN DO;
1063   3              DO I = 0 TO 3;
1064   4                 OLD$LAT(I) = OWN$SHIP$INFO.LAT(I);
1065   4                 OLD$LONG(I) = OWN$SHIP$INFO.LONG(I);
1066   4                 END;
1067   3              END;
1058   2        DO I = 0 TO LAST(ARRAY);
1069   3           IF ARRAY(I)
                   THEN DO;
1071   4               CALL CRT$PRINT$STRING(.TITLE$4);
1072   4               CALL SEND$CRLF;
1073   4               DO CASE I;
1074   5                  DO;
1075   6                    CALL GET$LAT(.OWN$SHIP$INFO.LAT);
1076   5                  END;
1077   5                  DO;
1078   6                    CALL GET$LONG(.OWN$SHIP$INFO.LONG);
1079   5                  END;
1080   5                  DO;
1081   5                    CALL GET$COURSE$BRG(0, .OWN$SHIP(J).CRS);
1082   6                    TEMP = FP$FORMAT(.OWN$SHIP(J).CRS, .CRS$STRING,
             -      3, 1);
1083   6                  END;
1084   5                  DO;
```

```
1085   6                 CALL GET$SPEED(.OWN$SHIP(J).SPD);
1086   6                 TEMP = FP$FORMAT(.OWN$SHIP(J).SPD, .SPD$STRING,
          -     2, 1);
1087   6               END;
1088   5               END;      /* CASE */
1089   4             END;        /* IF THEN */
1090   3           ELSE DO;
1091   4             DO CASE I;
1092   5               DO;
1093   6                 DO K = 0 TO 3;
1094   7                   OWN$SHIP(J).Y(K) = OWN$SHIP(LAST$INFO).Y(K);
1095   7                   END;
1096   6               END;
1097   5               DO;
1098   6                 DO K = 0 TO 3;
1099   7                   OWN$SHIP(J).X(K) = OWN$SHIP(LAST$INFO).X(K);
1100   7                   END;
1101   6               END;
1102   5               DO;
1103   6                 DO K = 0 TO 3;
1104   7                   OWN$SHIP(J).CRS(K) = OWN$SHIP(LAST$INFO).CRS
          -     (K);
1105   7                   END;
1106   6               END;
1107   5               DO;
1108   6                 DO K = 0 TO 3;
1109   7                   OWN$SHIP(J).SPD(K) = OWN$SHIP(LAST$INFO).SPD
          -     (K);
1110   7                   END;
1111   6               END;
```

```
1112   5                   END;       /* CASE */
1113   4                 END;         /* ELSE */
1114   3               END;           /* END DO */

1115   2             IF ARRAY(2) OR ARRAY(3)
                     THEN DO;
1117   3                 DO K = 0 TO 14;
1118   4                   IF CONTACT$INFO(K).DESIG <> 00H
                           THEN CONTACT$INFO(K).OS$POINTER = CONTACT$INFO(K).
            -   POINTER;
1120   4                   END;
1121   3                 END;
1122   2             IF ARRAY(0) OR ARRAY(1)
                     THEN DO;
1124   3                 CALL CONV$XY(.OWN$SHIP$INFO.LAT, .OWN$SHIP$INFO.LONG,
                                  .OWN$SHIP(J).X, .OWN$SHIP(J).Y);
1125   3                 CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LAT, .LAT$STRING,
            -    0);
1126   3                 CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LONG, .LONG$STRIN
            -   G, 1);
1127   3                 CALL PRINT$LAT$LONG(.LAT$STRING, .LONG$STRING);
1128   3                 CALL CONV$XY(.OLD$LAT, .OLD$LONG, .X$DELTA, .Y$DELTA)
            -   ;
1129   3                 CALL FSUB(.OWN$SHIP(J).X, .X$DELTA, .X$DELTA);
1130   3                 CALL FSUB(.OWN$SHIP(J).Y, .Y$DELTA, .Y$DELTA);
1131   3                 CALL TRANSLATE(.X$DELTA, .Y$DELTA, 0);
1132   3                 CALL CLEAR$STRUCTURES;
1133   3                 CALL SET$WINDOW;
1134   3                 CALL PUT$OS$CENTER;
1135   3                 CALL DRAW$EVERYTHING;
```

```
1136   3              CALL DISPLAY$PLASMA$SCALE;
1137   3              END;
1138   2          IF ARRAY(2) THEN CALL PRINT$COURSE(.CRS$STRING);
1140   2          IF ARRAY(3) THEN CALL PRINT$SPEED(.SPD$STRING);
1142   2          END OWN$SHIP$UPDATE;



                /*************************************************************
           -     ***************
                *
                * ORIGIN:
                *   THIS PROCEDURE IS USED TO MODIFY THE INFORMATION ABOUT TH
           -     E
                *   COORDINATE GRID ORIGIN.
                *
                *************************************************************
           -     ****************/
1143   1        ORIGIN: PROCEDURE PUBLIC;
1144   2          DCL OLD$LAT (4) BYTE,
                      OLD$LONG (4) BYTE,
                      DELTA$X (4) BYTE,
                      DELTA$Y (4) BYTE,
                      I BYTE;
1145   2          DO I = 0 TO 3;
1146   3             OLD$LAT(I) = SYSTEM.LAT(I);
1147   3             OLD$LONG(I) = SYSTEM.LONG(I);
1148   3             END;
1149   2          CALL CRT$PRINT$STRING(.TITLE$9);
```

```
1150   2          CALL SEND$CRLF;
1151   2          CALL GET$LAT(.SYSTEM.LAT);
1152   2          CALL CRT$PRINT$STRING(.TITLE$9);
1153   2          CALL SEND$CRLF;
1154   2          CALL GET$LONG(.SYSTEM.LONG);
1155   2          CALL CONV$XY(.OLD$LAT, .OLD$LONG, .DELTA$X, .DELTA$Y);
1156   2          CALL TRANSLATE(.DELTA$X, .DELTA$Y, 1);
1157   2          CALL CLEAR$STRUCTURES;
1158   2          CALL SET$WINDOW;
1159   2          CALL PUT$OS$CENTER;
1160   2          CALL DRAW$EVERYTHING;
1161   2          CALL DISPLAY$PLASMA$SCALE;
1162   2          END ORIGIN;


               /****************************************************************
               ***************
               *
               * WIND:
               *   THIS PROCEDURE IS USED TO GET INFORMATION ABOUT THE WIND.
               *
               *****************************************************************
               ****************/
1163   1       WIND: PROCEDURE PUBLIC;
1164   2          DCL BUFFER(7) BYTE,
                  (OK, TEMP) BYTE;
1165   2          DCL FP$360 (4) BYTE DATA (0DFH,0FFH,0B3H,043H);
1166   2          BUFFER(0) = 4;
1167   2          BUFFER(1) = 3;
```

```
1168   2        BUFFER(6) = 0;
1169   2        OK = 0;
1170   2        DO WHILE OK = 0;
1171   3           CALL CRT$PRINT$STRING(.TITLE$A);
1172   3           CALL SEND$CRLF;
1173   3           CALL CRT$PRINT$STRING(.('ENTER THE WIND DIRECTION AS RE
          -   QUESTED:$$'));
1174   3           CALL SEND$CRLF;
1175   3           TEMP = 0;
1176   3           DO WHILE TEMP = 0;
1177   4              CALL CRT$PRINT$STRING(.('DEGREES: $$'));
1178   4              CALL PUT$NUMBER$BUFFER(3, .BUFFER(2));
1179   4              CALL CRT$WRITE('.');
1180   4              CALL PUT$NUMBER$BUFFER(1, .BUFFER(5));
1181   4              CALL ASCII$TO$FLOAT(.BUFFER, 7, .SYSTEM.WIND$DIR);
1182   4              TEMP = CHECK$FP$VALUE(.SYSTEM.WIND$DIR, .FP$360);
1183   4              END;
1184   3           CALL SEND$CRLF;
1185   3           OK = CHECK$INPUT;
1186   3           CALL CLEAR$LOW$SCREEN;
1187   3           END;
1188   2        BUFFER(0) = 3;
1189   2        BUFFER(1) = 2;
1190   2        BUFFER(5) = 0;
1191   2        OK = 0;
1192   2        DO WHILE OK =0;
1193   3           CALL CRT$PRINT$STRING(.TITLE$A);
1194   3           CALL SEND$CRLF;
1195   3           CALL CRT$PRINT$STRING(.('ENTER THE WIND SPEED AS REQUES
          -   TED:$$'));
```

```
3           CALL SEND$CRLF;
3           CALL CRT$PRINT$STRING(.('KNOTS: $$'));
3           CALL PUT$NUMBER$BUFFER(2, .BUFFER(2));
3           CALL CRT$WRITE('.');
3           CALL PUT$NUMBER$BUFFER(1, .BUFFER(4));
3           CALL ASCII$TO$FLOAT(.BUFFER, 6, .SYSTEM.WIND$SPD);
3           CALL SEND$CRLF;
3           OK = CHECK$INPUT;
3           CALL CLEAR$LOW$SCREEN;
3           END;
2         END WIND;


        /*************************************************************
   -     ******************
        *
        * SCALE:
        *   THIS PROCEDURE IS USED TO UPDATE THE GRAPHICS SCALE VALUE
   -     .
        *
        *************************************************************
   -     *****************/
1       SCALE: PROCEDURE PUBLIC;
2         CALL CRT$PRINT$STRING(.TITLE$B);
2         CALL SEND$CRLF;
2         CALL GET$SCALE(.SYSTEM.SCALE);
2         CALL CLEAR$STRUCTURES;
2         CALL SET$WINDOW;
2         CALL PUT$OS$CENTER;
```

```
1214   2          CALL DRAW$EVERYTHING;
1215   2          CALL DISPLAY$PLASMA$SCALE;
1216   2          END SCALE;


                /*************************************************************
            -    *************
                *
                * GET$SAFE$RNG:
                *    THIS PROCEDURE IS USED TO OBTAIN THE VALUE OF SAFE CPA RA
            -    NGE USED TO
                *    WARN THE OPERATOR THAT A CONTACT WILL BE IN COLLISION.
                *
                *************************************************************
            -    *************/
1217   1        GET$SAFE$RNG: PROCEDURE PUBLIC;
1218   2          DCL BUFFER (7) BYTE,
                    (OK, TEMP, TEMP1) BYTE;
1219   2          DCL LOW$BOUND (4) BYTE DATA (008H, 03CH, 0CAH, 03CH), /* 0
            -    .024686827 */
                      HIGH$BOUND (4) BYTE DATA (000H, 000H, 000H, 03FH),/* 0
            -    .5 */
                      PP$2000 (4) BYTE DATA (0E4H, 02BH, 0FDH, 044H);   /* 2
            -    025.3716 */
1220   2          DCL M0 (*) BYTE DATA ('ENTER THE SAFE C.P.A. RANGE AS REQU
            -    ESTED:$$'),
                      M1 (*) BYTE DATA ('YARDS:  $$');

1221   2          OK = 0;
```

```
1222   2          DO WHILE OK = 0;
1223   3             CALL CRT$PRINT$STRING(.TITLE$D);
1224   3             CALL SEND$CRLF;
1225   3             CALL CRT$PRINT$STRING(.M0);
1226   3             CALL SEND$CRLF;
1227   3             TEMP, TEMP1 = 0;
1228   3             DO WHILE (TEMP = 0) OR (TEMP1 = 0);
1229   4                CALL CRT$PRINT$STRING(.M1);
1230   4                BUFFER(0), BUFFER(1) = 4;
1231   4                BUFFER(6) = 0;
1232   4                CALL PUT$NUMBER$BUFFER(4, .BUFFER(2));
1233   4                CALL ASCII$TO$FLOAT(.BUFFER, 7, .SAFE$RNG);
1234   4                CALL FDIV(.SAFE$RNG, .FP$2000, .SAFE$RNG);
1235   4                TEMP = CHECK$FP$VALUE(.SAFE$RNG, .HIGH$BOUND);
1236   4                IF TEMP <> 0
                             THEN TEMP1 = CHECK$FP$VALUE(.LOW$BOUND, .SAFE$RNG
               -   );
1238   4                END;
1239   3             CALL SEND$CRLF;
1240   3             OK = CHECK$INPUT;
1241   3             CALL CLEAR$LOW$SCREEN;
1242   3             END;
1243   2          END GET$SAFE$RNG;


                   /****************************************************************
               -    *************
                    *
                    * INPUT$TIME:
```

```
                *   THIS PROCEDURE IS USED TO AITER ALL VALUES CONCERNING WIT
           -     H TIME:
                *       - TIME ZONE NUMBER.
                *       - SYSTEM CLOCK TIME.
                *       - TIME BETWEEN UPDATES OF OWN SHIP POSITIONS.
                *
                * USAGE:
                *   TYPED PROCEDURE. IF THE TIME BETWEEN UPDATES IS CHANGED (
           -     DEFAULT 180 SEC-
                *   ONDS) THEN THE NEW VALUE IS RETURNED, OTHERWISE A VALUE O
           -     F ZERO IS RETUR-
                *   NED. THE VALUE MUST BE BETWEEN 250 AND 15.
                *
                *************************************************************
           -     ****************/
1244   1        INPUT$TIME: PROCEDURE BYTE PUBLIC;
1245   2           DCL ARRAY(3) BYTE,
                       VALUE ADDRESS,
                       (I, OK, TEMP) BYTE;
1246   2           DCL M0 (*) BYTE DATA ('TIME ZONE NUMBER?       (Y/N)  $$'),
                       M1 (*) BYTE DATA ('SYSTEM CLOCK VALUE?     (Y/N)  $$'),
                       M2 (*) BYTE DATA ('TIME BETWEEN UPDATES?   (Y/N)  $$'),
                       M3 (*) BYTE DATA ('ENTER TIME BETWEEN UPDATES AS REQUE
           -     STED:$$'),
                       M4 (*) BYTE DATA ('SECONDS:  $$'),
                       M5 (*) BYTE DATA (' *** BAD FORMAT ***$$');

1247   2           OK = 0;
1248   2           DO WHILE OK = 0;
1249   3              CALL CRT$PRINT$STRING(.TITLE$6);
```

```
1250   3           CALL SEND$CRLF;
1251   3           CALL CRT$PRINT$STRING(.MSG$1);
1252   3           CALL SEND$CRLF;
1253   3           CALL CRT$PRINT$STRING(.M0);
1254   3           ARRAY(0) = CHECK$YES$NO;
1255   3           CALL SEND$SPACE(13);
1256   3           CALL CRT$PRINT$STRING(.M1);
1257   3           ARRAY(1) = CHECK$YES$NO;
1258   3           CALL SEND$CRLF;
1259   3           CALL CRT$PRINT$STRING(.M2);
1260   3           ARRAY(2) = CHECK$YES$NO;
1261   3           CALL SEND$CRLF;
1262   3           OK = CHECK$INPUT;
1263   3           CALL CLEAR$LOW$SCREEN;
1264   3           END;
1265   2         DO I = 0 TO 2;
1266   3           IF ARRAY(I)
                   THEN DO;
1268   4              CALL CRT$PRINT$STRING(.TITLE$6);
1269   4              CALL SEND$CRLF;
1270   4              DO CASE I;
1271   5                 DO;
1272   6                   CALL GET$TIME$ZONE(.SYSTEM.NUM$ZONE);
1273   6                   CALL PRINT$TIME$ZONE(.SYSTEM.NUM$ZONE);
1274   6                 END;
1275   5                 DO;
1276   6                   CALL INITIATE$TIME;
1277   6                 END;
1278   5                 DO;
1279   6                   OK = 0;
```

```
1280   6                   DO WHILE OK = 0;
1281   7                      CALL CRT$PRINT$STRING(.M3);
1282   7                      CALL SEND$CRLF;
1283   7                      VALUE = 0;
1284   7                      DO WHILE (VALUE > 250) OR (VALUE < 15);
1285   8                         CALL CRT$PRINT$STRING(.M4);
1286   8                         VALUE = GET$ADDRESS(3);
1287   8                         IF (VALUE > 250) OR (VALUE < 15)
                                 THEN DO;
1289   9                              CALL CRT$PRINT$STRING(.M5);
1290   9                              CALL SEND$BEL;
1291   9                              CALL SEND$CR;
1292   9                              CALL SEND$SUB;
1293   9                              END;
1294   8                         ELSE DO;
1295   9                              CALL CRT$WRITE(17H);   /* ERASE TO
         -    END OF LINE */
1296   9                              CALL SEND$CRLF;
1297   9                              END;
1298   8                         END;
1299   7                      OK = CHECK$INPUT;
1300   7                      CALL CLEAR$LOW$SCREEN;
1301   7                      END;
1302   6                END;
1303   5                END;           /* END CASE */
1304   4             END;              /* END THEN DO */
1305   3         END;                  /* END DO */
1306   2      IF ARRAY(2) THEN RETURN LOW(VALUE);
1308   2                  ELSE RETURN 0;
1309   2      END INPUT$TIME;
```

```
1310   1          END EXECUTIVE$CMDS;


MODULE INFORMATION:

     CODE AREA SIZE     = 2944H  10564D
     VARIABLE AREA SIZE = 1BE9H   7145D
     MAXIMUM STACK SIZE = 0008H      8D
     1930 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
 519   5                  DO;               /* CASE 39H */
 520   5                    CALL ORIGIN;
 521   6                  END;

 522   5                  END;              /* END CASE */
 523   4              END;                  /* END THEN DO */
 524   3            END;                    /* END DO FOREVER */


 525   2          END EXECUTIVE;


 526   1       END MAIN$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 06C6H   1734D
     VARIABLE AREA SIZE = 002FH     47D
     MAXIMUM STACK SIZE = 0008H      8D
     1053 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
 493   6                   ELSE CALL NO$CONTACT;
 494   6               END;

 495   5               DO;              /* CASE 35H */
 496   6                 IF (SYSTEM.NUMCTS > 6)
                           THEN CALL SWAP$CONTACTS;
 498   6                   ELSE DO;
 499   7                       IF SYSTEM.NUMCTS <> 0
                                  THEN CALL NOT$ENOUGH$CONTACTS;
 501   7                          ELSE CALL NO$CONTACT;
 502   7                       END;
 503   6               END;

 504   5               DO;              /* CASE 36H */
 505   6                 CALL WIND;
 506   6                 WIND$FLAG = TRUE;
 507   6               END;

 508   5               DO;              /* CASE 37H */
 509   6                 IF SYSTEM.NUMCTS < 15
                         THEN DO;
 511   7                     CALL CREATE;
 512   7                     SYSTEM.NUMCTS = SYSTEM.NUMCTS + 1;
 513   7                     END;
 514   6                 ELSE CALL TOO$MANY$CONTACTS;
 515   6               END;

 516   5               DO;              /* CASE 38H */
 517   6                 CALL OWN$SHIP$UPDATE;
 518   6               END;
 469   6              END;

 470   5              DO;              /* CASE 2FH */
 471   6                IF SYSTEM.NUMCTS > 0
                            THEN CALL UPDATE;
 473   6                    ELSE CALL NO$CONTACT;
 474   6              END;

 475   5              DO;                 /* CASE 30H */
 476   6                CALL GET$SAFE$RNG;
 477   5              END;

 478   5              DO;                 /* CASE 31H */
 479   5                IF SYSTEM.NUMCTS > 0
                            THEN CALL REDESIGNATE;
 481   5                    ELSE CALL NO$CONTACT;
 482   6              END;

 483   5              DO;                 /* CASE 32H */
 484   6                IF(TEMP:= INPUT$TIME) <> 0
                            THEN TIME$LIMIT = TEMP;
 486   6              END;

 487   5              DO;                 /* CASE 33H */
 488   6                CALL SCALE;
 489   6              END;

 490   5              DO;                 /* CASE 34H */
 491   6                IF SYSTEM.NUMCTS > 0
                            THEN CALL REMOVE;
```

```
449   5                   ;             /* CASE 12H */

450   5                   ;             /* CASE 13H */

451   5                   ;             /* CASE 14H */

452   5                   ;             /* CASE 15H */

453   5                   ;             /* CASE 16H */

454   5                   ;             /* CASE 17H */

455   5                   ;             /* CASE 18H */

456   5                   DO;           /* CASE 19H */
457   6                     IF SYSTEM.NUMCTS > 0
                              THEN CALL DISPLAY$SYSTEM;
459   6                       ELSE CALL NO$CONTACT;
460   6                   END;

461   5                   END;          /* END CASE */
462   4               CALL PRINT$MODE(.INPUT);
463   4               END;              /* END THEN DO */

464   3             IF (COMMAND > INPUT$LOWER$LIMIT) AND (COMMAND < INPUT$U
        -       PPER$LIMIT)
                    THEN DO;            /* INPUT COMMAND DETECTED */
466   4                 DO CASE (COMMAND - 2EH);
467   5                   DO;           /* CASE 2EH */
468   6                     CALL REORIENT$PS;
```

```
 429   5              ;              /* CASE 08H */

 430   5              ;              /* CASE 09H */

 431   5              ;              /* CASE 0AH */

 432   5              DO;            /* CASE 0BH */
 433   6                IF SYSTEM.NUMCTS > 0
                             THEN CALL DISPLAY$CONTACT$INFO;
 435   6                     ELSE CALL NO$CONTACT;
 436   6              END;

 437   5              DO;            /* CASE 0CH */
 438   5                CALL DISPLAY$ORIGIN;
 439   6              END;

 440   5              ;              /* CASE 0DH */

 441   5              DO;            /* CASE 0EH */
 442   6                CALL DISPLAY$SCALE;
 443   6              END;

 444   5              DO;            /* CASE 0FH */
 445   6                CALL DISPLAY$OWN$SHIP;
 446   6              END;

 447   5              ;              /* CASE 10H */

 448   5              ;              /* CASE 11H */
 409   3            IF (COMMAND < DISPLAY$UPPER$LIMIT) AND
                       (COMMAND <> 0)
                                    /* CHECK FOR DISPLAY TYPE COMMANDS */
                    THEN DO;
 411   4                  CALL PRINT$MODE(.DISPLAY);
 412   4                  DO CASE COMMAND;
 413   5                      ;               /* CASE 00H */

 414   5                      DO;             /* CASE 01H */
 415   6                        IF WIND$FLAG
                                    THEN CALL DISPLAY$WIND;
 417   6                            ELSE CALL NO$WIND;
 418   6                      END;

 419   5                      ;               /* CASE 02H */

 420   5                      ;               /* CASE 03H */

 421   5                      DO;             /* CASE 04H */
 422   6                        CALL DISPLAY$SAFE$ENG;
 423   6                      END;

 424   5                      ;               /* CASE 05H */

 425   5                      DO;             /* CASE 06H */
 426   6                        CALL DISPLAY$UPDATE$TIME(TIME$LIMIT);
 427   6                      END;

 428   5                      ;               /* CASE 07H */
 387   2          TIME$STEP = 00H;
 388   2          TIME$LIMIT = 100;
 389   2          WIND$FLAG = FALSE;
 390   2          CALL CLEAR$STRUCTURES;
 391   2          CALL INITIALIZE$PLASMA;
 392   2          CALL GET$SYSTEM$PARAMETERS;
 393   2          CALL SET$WINDOW;
 394   2          CALL CLEAR$PLASMA;
 395   2          CALL DISPLAY$PLASMA$SCALE;
 396   2          CALL PUT$OS$CENTER;
 397   2          CALL PLASMA$OS;


 398   2          DO FOREVER;
 399   3             IF SEC$TIME          /* A SECOND HAS ELAPSED. DISPLAY ACT
               -  UAL TIME */
                       THEN DO;
 401   4                  SEC$TIME = FALSE;
 402   4                  CALL ACTUAL$TIME;
 403   4                  CALL PRINT$TIME(.TIME$BUFFER);
 404   4                  CALL CRT$WRITE(PROMPT);
 405   4                  END;

 406   3             IF TIME$STEP >= TIME$LIMIT
                          THEN CALL MOVE$OWN$SHIP;    /* TIME TO UPDATE OWN SH
               -  IP POSITION */

 408   3             COMMAND = CRT$TRY$READ;       /* CHECK FOR INPUT FROM KE
               -  YBOARD */
 375   2          CALL FADD(.OWN$SHIP(LAST$INFO).Y, .DELTA$Y, .OWN$SHIP(POIN
             -   TER).Y);
 376   2          CALL CONV$LAT$LONG(.OWN$SHIP(POINTER).X, .OWN$SHIP(POINTER
             -   ).Y,                            .OWN$SHIP$INFO.LAT, .OWN$SHIP$INFO.LONG
             -   );
 377   2          DO I = 0 TO 3;
 378   3             OWN$SHIP(POINTER).CRS(I) = OWN$SHIP(LAST$INFO).CRS(I);
 379   3             OWN$SHIP(POINTER).SPD(I) = OWN$SHIP(LAST$INFO).SPD(I);
 380   3             END;
                     /* DISPLAY ACTUAL VALUES. */
 381   2          CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LAT, .LAT$STRING, 0);
 382   2          CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LONG, .LONG$STRING, 1)
             -   ;
 383   2          CALL PRINT$LAT$LONG(.LAT$STRING, .LONG$STRING);
                     /* DRAW NEW POSITION IN PLASMA DISPLAY. */
 384   2          CALL PLASMA$OS;
                     /* ALL DONE. RETURN */
 385   2          END MOVE$OWN$SHIP;


                                     /***   EXECUTIVE:   ***/


 386   1         EXECUTIVE: DO;
```

```
354   2          IF OWN$SHIP$INFO.POINTER = 30
                 THEN DO;
356   3               OWN$SHIP$INFO.POINTER = 0;
357   3               OWN$SHIP$INFO.FLAG = TRUE;
358   3               END;
359   2          LAST$INFO = POINTER;
360   2          POINTER = OWN$SHIP$INFO.POINTER;
361   2          OWN$SHIP(POINTER).TIME(0) = OWN$SHIP(LAST$INFO).TIME(0) +
          -    H;
362   2          OWN$SHIP(POINTER).TIME(1) = OWN$SHIP(LAST$INFO).TIME(1) +
          -    M;
363   2          OWN$SHIP(POINTER).TIME(2) = OWN$SHIP(LAST$INFO).TIME(2) +
          -    S;
364   2          DO WHILE OWN$SHIP(POINTER).TIME(2) >= 60;
365   3             OWN$SHIP(POINTER).TIME(2) = OWN$SHIP(POINTER).TIME(2) -
          -     60;
366   3             OWN$SHIP(POINTER).TIME(1) = OWN$SHIP(POINTER).TIME(1) +
          -     1;
367   3             END;
368   2          DO WHILE OWN$SHIP(POINTER).TIME(1) >= 60;
369   3             OWN$SHIP(POINTER).TIME(1) = OWN$SHIP(POINTER).TIME(1) -
          -     60;
370   3             OWN$SHIP(POINTER).TIME(0) = OWN$SHIP(POINTER).TIME(0) +
          -     1;
371   3             END;
372   2          IF OWN$SHIP(POINTER).TIME(0) >= 24
                    THEN OWN$SHIP(POINTER).TIME(0) = OWN$SHIP(POINTER).TIME
          -    (0) - 24;
374   2          CALL FADD(.OWN$SHIP(LAST$INFO).X, .DELTA$X, .OWN$SHIP(POIN
          -    TER).X);
```

```
                -     0.0174532925 */

                         /* SAVE TIME OF CALL */
 334   2              H, M, S = 00H;
 335   2              S = TIME$STEP;
 336   2              TIME$STEP = 00H;
 337   2              TIME(1), TIME(2), TIME(3) = 00H;
 338   2              TIME(0) = LOW(S);
 339   2              IF S > 255
                         THEN TIME(1) = HIGH(S);
                         /* FIND INTERVAL OF TIME PAST */
 341   2              DO WHILE S >= 60;
 342   3                 S = S - 60;
 343   3                 M = M + 1;
 344   3                 END;
                         /* CONVERT TIME TO FP FORMAT */
 345   2              CALL FLTDS(.TIME, .TIME$FLOAT);
                         /* CONVERT SPEED IN KNOTS INTO MILES/SECONDS */
 346   2              POINTER = OWN$SHIP$INFO.POINTER;
 347   2              CALL FDIV(.OWN$SHIP(POINTER).SPD, .FP$3600, .SPEED);
                         /* CONVERT COURSE TO ANGLE IN RADIANS */
 348   2              CALL FMUL(.OWN$SHIP(POINTER).CRS, .DEG$TO$RAD, .COURSE);
                         /* GET SINE AND COSINE VALUES */
 349   2              CALL COS$SIN(.COURSE, .COS, .SIN);
                         /* FIND VARIATIONS IN X AND Y PARAMETERS */
 350   2              CALL FMUL(.TIME$FLOAT, .SPEED, .DISTANCE);
 351   2              CALL FMUL(.DISTANCE, .SIN, .DELTA$X);
 352   2              CALL FMUL(.DISTANCE, .COS, .DELTA$Y);
                         /* UPDATE OWN SHIP VALUES */
 353   2              OWN$SHIP$INFO.POINTER = OWN$SHIP$INFO.POINTER + 1;
```

```
                   /************************************************************
                -   *****************
                   *
                   * MOVE$OWN$SHIP:
                   *   THIS PROCEDURE IS EXECUTED EACH TIME A PREDETERMINED PERI
                -   OD OF TIME ELAP-
                   *   SES. IT IS USED TO CALCULATE THE MOVEMENTS OF THE SHIP DU
                -   RING THAT PERIOD.
                   *
                   ************************************************************
                -   ****************/
 331   1           MOVE$OWN$SHIP: PROCEDURE PUBLIC;
 332   2              DCL DELTA$X (4) BYTE,
                          DELTA$Y (4) BYTE,
                          COS (4) BYTE,
                          SIN (4) BYTE,
                          TIME (4) BYTE,
                          TIME$FLOAT (4) BYTE,
                          SPEED (4) BYTE,
                          COURSE (4) BYTE,
                          DISTANCE (4) BYTE,
                          S ADDRESS,
                          (I, POINTER, LAST$INFO, H, M) BYTE;

 333   2              DCL FP$3600 (4) BYTE DATA (00H, 00H, 61H, 45H),         /*
                -    3600.00 */
                          DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH);  /*
 315    2          CALL SET$LOW$HOME;
 316    2          CALL CRT$PRINT$STRING(.MSG);
 317    2          CALL SEND$CRLF;
 318    2          CALL SEND$CRLF;
 319    2          CALL CHECK$GO$KEY;
 320    2          CALL CLEAR$LOW$SCREEN;
 321    2          END NOT$ENOUGH$CONTACTS;


               /**************************************************************
           -    ****************
               *
               * TOO$MANY$CONTACTS:
               *    THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE SIST
           -    EM CAN NOT ACCEPT
               *    ANY MORE NEW CONTACTS.
               *
               ***************************************************************
           -    ****************/
 322    1      TOO$MANY$CONTACTS: PROCEDURE;
 323    2          DCL MSG (*) BYTE DATA
                          ('SYSTEM ALREADY MAINTAINS 15 CONTACTS.$$');
 324    2          CALL SET$LOW$HOME;
 325    2          CALL CRT$PRINT$STRING(.MSG);
 326    2          CALL SEND$CRLF;
 327    2          CALL SEND$CRLF;
 328    2          CALL CHECK$GO$KEY;
 329    2          CALL CLEAR$LOW$SCREEN;
 330    2          END TOO$MANY$CONTACTS;
```

```
304   1      NO$CONTACT: PROCEDURE;
305   2         DCL MSG (*) BYTE DATA
                    ('NO CONTACTS ARE BEING MAINTAINED BY THE SYSTEM.$
           -  $');
306   2         CALL SET$LOW$HOME;
307   2         CALL CRT$PRINT$STRING(.MSG);
308   2         CALL SEND$CRLF;
309   2         CALL SEND$CRLF;
310   2         CALL CHECK$GO$KEY;
311   2         CALL CLEAR$LOW$SCREEN;
312   2         END NO$CONTACT;


             /*************************************************************
           -  ***************
             *
             * NOT$ENOUGH$CONTACTS:
             *   THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE NUMB
           -  ER OF CONTACTS
             *   PRESENTLY AT THE SYSTEM, IS NOT ENOUGH TO ALLOW SWAPING O
           -  F CONTACTS NOT
             *   IN THE SCREEN.
             *
             **************************************************************
           -  **************/
313   1      NOT$ENOUGH$CONTACTS: PROCEDURE;
314   2         DCL MSG (*) BYTE DATA
                    ('ALL CONTACTS IN THE SYSTEM ARE ALREADY DISPLAYED
           -  .$$');
```

```
                   * NO$WIND:
                   *   THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT NO WIND
                -   INFORMATION
                   *   EXISTS IN THE SYSTEM.
                   *
                   ************************************************************
                -   *************/
 295    1          NO$WIND: PROCEDURE;
 296    2             DCL MSG (*) BYTE DATA
                            ('NO WIND INFORMATION AVAILABLE.$$');
 297    2             CALL SET$LOW$HOME;
 298    2             CALL CRT$PRINT$STRING(.MSG);
 299    2             CALL SEND$CRLF;
 300    2             CALL SEND$CRLF;
 301    2             CALL CHECK$GO$KEY;
 302    2             CALL CLEAR$LOW$SCREEN;
 303    2             END NO$WIND;


                   /***********************************************************
                -   **************
                   *
                   * NO$CONTACT:
                   *   THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE SYST
                -   EM IS NOT CUR-
                   *   RENTLY MAINTAINING ANY CONTACT.
                   *
                   ************************************************************
                -   **************/
```

```
                        PROCEDURE EXTERNAL;
 291   2                END;


                                                  /***   DECLARATIONS:   ***/

 292   1              DCL TRUE LIT '0FFH',
                          FALSE LIT '00H',
                          FOREVER LIT 'WHILE TRUE',
                          PROMPT LIT '025H',
                          DISPLAY$UPPER$LIMIT LIT '01AH',
                          INPUT$LOWER$LIMIT LIT '02DH',
                          INPUT$UPPER$LIMIT LIT '03AH';

 293   1              DCL DISPLAY (*) BYTE DATA ('DISPLAY$$'),
                          INPUT   (*) BYTE DATA (' INPUT $$');

 294   1              DCL TIME$LIMIT BYTE,
                          TEMP BYTE,
                          WIND$FLAG BYTE,
                          COMMAND BYTE;

                      /************************************************************
                       *************
                      *
 276   1      INITIALIZE$PLASMA:
                 PROCEDURE EXTERNAL;
 277   2         END;


 278   1      SET$WINDOW:
                 PROCEDURE EXTERNAL;
 279   2         END;

 280   1      CLEAR$STRUCTURES:
                 PROCEDURE EXTERNAL;
 281   2         END;

 282   1      PUT$OS$CENTER:
                 PROCEDURE EXTERNAL;
 283   2         END;

 284   1      PLASMA$OS:
                 PROCEDURE EXTERNAL;
 285   2         END;

 286   1      DRAW$EVERYTHING:
                 PROCEDURE EXTERNAL;
 287   2         END;

 288   1      DISPLAY$PLASMA$SCALE:
                 PROCEDURE EXTERNAL;
 289   2         END;

 290   1      REORIENT$PS:
```

```
PL/M-80 COMPILER                                           20 MAR 81  PAGE   2

 260   2            END;

 261   1          DISPLAY$SCALE:
                    PROCEDURE EXTERNAL;
 262   2            END;

 263   1          DISPLAY$OWN$SHIP:
                    PROCEDURE EXTERNAL;
 264   2            END;

 265   1          DISPLAY$SAFE$RNG:
                    PROCEDURE EXTERNAL;
 266   2            END;

 267   1          DISPLAY$SYSTEM:
                    PROCEDURE EXTERNAL;
 268   2            END;

 269   1          DISPLAY$UPDATE$TIME:
                    PROCEDURE (A) EXTERNAL;
 270   2            DECLARE A BYTE; END;

 272   1          GET$SAFE$RNG:
                    PROCEDURE EXTERNAL;
 273   2            END;

 274   1          CLEAR$PLASMA:
                    PROCEDURE EXTERNAL;
 275   2            END;
```

```
PL/M-80 COMPILER                                          20 MAR 81  PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MAINMODULE
OBJECT MODULE PLACED IN ANAMOD.OBJ
COMPILER INVOKED BY:  :F1:PLM80 ANAMOD.SRC PAGELENGTH(35) PAGEWIDTH(75) DAT
                    -E(20 MAR 81)



   1          MAIN$MODULE: DO;
                                   /***    EXTERNALS:   ***/


              $NOLIST

 252   1      CONV$LAT$LONG:
                 PROCEDURE (A,B,C,D) EXTERNAL;
 253   2         DCL (A,B,C,D) ADDRESS; END;

 255   1      DISPLAY$WIND:
                 PROCEDURE EXTERNAL;
 256   2         END;

 257   1      DISPLAY$CONTACT$INFO:
                 PROCEDURE EXTERNAL;
 258   2         END;

 259   1      DISPLAY$ORIGIN:
                 PROCEDURE EXTERNAL;
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE AAWGFCS
OBJECT MODULE PLACED IN AAW.OBJ
COMPILER INVOKED BY:  :F1:PLM80 AAW.SRC PAGELENGTH(33) PAGEWIDTH(75) DATE(2
                    -0 MAR 81)


   1            AAWSGFCS: DO;

                /***   EXTERNALS   ***/

   2   1        CRT$WRITE:
                  PROCEDURE (CHAR) EXTERNAL;
   3   2          DECLARE CHAR BYTE; END;

   5   1        CRT$PRINT$STRING:
                  PROCEDURE (A) EXTERNAL;
   6   2          DECLARE A ADDRESS; END;

   8   1        SEND$CRLF:
                  PROCEDURE EXTERNAL;
   9   2          END;

  10   1        SEND$SPACE:
                  PROCEDURE (NUM) EXTERNAL;
  11   2          DECLARE NUM BYTE; END;

  13   1        FADD:
                  PROCEDURE (A,B,C) EXTERNAL;
  14   2          DECLARE (A,B,C) ADDRESS; END;
```

```
 16    1        FSUB:
                  PROCEDURE (A,B,C) EXTERNAL;
 17    2          DECLARE (A,B,C) ADDRESS; END;
 19    1        FMUL:
                  PROCEDURE (A,B,C) EXTERNAL;
 20    2          DECLARE (A,B,C) ADDRESS; END;
 22    1        FDIV:
                  PROCEDURE (A,B,C) EXTERNAL;
 23    2          DECLARE (A,B,C) ADDRESS; END;
 25    1        FSQR:
                  PROCEDURE (A,B) EXTERNAL;
 26    2          DECLARE (A,B) ADDRESS; END;
 28    1        FSQRT:
                  PROCEDURE (A,B) EXTERNAL;
 29    2          DECLARE (A,B) ADDRESS; END;
 31    1        FCMPR:
                  PROCEDURE (A,B,C) BYTE EXTERNAL;
 32    2          DECLARE (A,B,C) ADDRESS; END;
 34    1        FZTST:
                  PROCEDURE (A,B) BYTE EXTERNAL;
 35    2          DECLARE (A,B) ADDRESS; END;
 37    1        EXCH:
                  PROCEDURE (A,B) EXTERNAL;
 38    2          DECLARE (A,B) ADDRESS; END;
 40    1        COSSSIN:
                  PROCEDURE (A,B,C) EXTERNAL;
 41    2          DECLARE (A,B,C) ADDRESS; END;
 43    1        ARCSTAN:
                  PROCEDURE (A,B,C) EXTERNAL;
 44    2          DECLARE (A,B,C) ADDRESS; END;
 46   1        FLOAT$TO$ASCII:
                 PROCEDURE (A,B,C) EXTERNAL;
 47   2          DECLARE (A,B,C) ADDRESS; END;
 49   1        INIT$FP:
                 PROCEDURE EXTERNAL;
 5Ø   2          END;
 51   1        FP$FORMAT:
                 PROCEDURE (A, B, C, D) BYTE EXTERNAL;
 52   2          DECLARE (A,B) ADDRESS, (C,D) BYTE; END;

 54   1        WRITE:
                 PROCEDURE (AFT,BUFFER,LENGTH,STATUS) EXTERNAL;
 55   2          DECLARE (AFT,BUFFER,LENGTH,STATUS) ADDRESS;
 56   2          END WRITE;
 57   1        EXIT:
                 PROCEDURE EXTERNAL;
 58   2          END EXIT;

               /*****   DECLERATIONS   ***/

 59   1        DECLARE LIT LITERALLY 'LITERALLY',
                       DCL LIT 'DECLARE';
 6Ø   1        DCL BUFFER (128) BYTE;
 61   1        DCL STATUS ADDRESS;
 62   1        DCL CRLF(2) BYTE DATA (ØDH, ØAH);
 63   1        DCL BIN$RNG BYTE AT (ØF7Ø5H),
                   BIN$FLV BYTE AT (ØF7Ø3H),
                   BIN$BRG BYTE AT (ØF7Ø4H),
                   BIN$VX BYTE AT (ØF7ØØH),
                   BIN$VY BYTE AT (ØF7Ø1H),
                    BIN$VZ BYTE AT (0F702H);

  64    1       DCL FRNG (4) BYTE,
                    FELV (4) BYTE,
                    FBRG (4) BYTE,
                    FVX (4) BYTE,
                    FVY (4) BYTE,
                    FVZ (4) BYTE;

  65    1       DCL FXO (4) BYTE,
                    FXT (4) BYTE,
                    FYO (4) BYTE,
                    FYT (4) BYTE;

  66    1       DCL FSH (4) BYTE,
                    FSR (4) BYTE,
                    FCT (4) BYTE,
                    FBSTRU (4) BYTE,
                    FTSANG (4) BYTE,
                    COS$FBRG (4) BYTE,
                    SIN$FBRG (4) BYTE,
                    COS$FTSANG (4) BYTE,
                    SIN$FTSANG (4) BYTE;

  67    1       DCL FDRH (4) BYTE,
                    FRDES (4) BYTE,
                    FDR (4) BYTE,
                    FRDE (4) BYTE,
                    FDER (4) BYTE,
                    FDE (4) BYTE;
  68    1       DCL FCR (4) BYTE,
                    FCER(4) BYTE,
                    FCE (4) BYTE,
                    COS$FCE (4) BYTE,
                    SIN$FCE (4) BYTE,
                    FH (4) BYTE;

  69    1       DCL TF (4) BYTE;

  70    1       DCL PRNG (4) BYTE,
                    PELV (4) BYTE,
                    PBRG (4) BYTE,
                    PH (4) BYTE;

  71    1           DCL OBRG (4) BYTE,
                        OELV (4) BYTE;

  72    1           DCL OENBRG BYTE AT (0F700H),
                        OENELV BYTE AT (0F701H);

  73    1       DCL VALUE BYTE;

  74    1       DCL PLUS$SIGN LIT '02BH',
                    MINUS$SIGN LIT '02DH',
                    COLON LIT '03AH',
                    POINT LIT '02EH',
                    BLANK LIT '020H';

  75    1       DCL FSO (4) BYTE DATA(0FEH, 54H, 05H, 42H);     /* OWN SHIP SEE
```

```
                 -   FD */
 76    1         DCL FCOWN (4) BYTE DATA(ØDBH, ØFH, 49H, 3FH); /* OWN SHIP COU
                 -   RSE */
 77    1         DCL DEGSTO$RAL (4) BYTE DATA(Ø3F, ØFAH, ØEEH, Ø3CH); /*  Ø.Ø1
                 -   74532 */
 78    1         DCL RADSTOSDEG (4) BYTE DATA (32H, 2FH, 65H, 42H);
 79    1         DCL FP$360 (4) BYTE DATA (ØØH, ØØH, ØB4H, 43H),
                       FP$225 (4) BYTE DATA (ØØH, ØØH, 51H, 43H),
                       FP$180 (4) BYTE DATA (ØØH, ØØH, 34H, 43H),
                       FP$135 (4) BYTE DATA (ØØH, ØØH, Ø7H, 43H),
                       FP$Ø$819 (4) BYTE DATA (2FH, ØC1H, 51H, 3FH),
                       MINS1 (4) BYTE DATA (ØØH, ØØH, 8ØH, ØBFH),
                       FPSNG$135 (4) BYTE DATA (ØØH, ØØH, Ø7H, ØCEH);

 80    1           DCL FP$767$39 (4) BYTE DATA (Ø32H, ØD9H, Ø44H, Ø44H);
 81    1           DCL FP$2$708 (4) BYTE DATA (ØD5H, Ø6AH, Ø35H, Ø2FH);
 82    1           DCL FP$2$83 (4) BYTE DATA (ØD3H, Ø6AH, Ø35H, Ø4ØH);
 83    1           DCL FP$3$93 (4) BYTE DATA (ØCFH, ØF7H, Ø7BH, Ø4ØH);
 84    1           DCL FP$1$33 (4) BYTE DATA (Ø95H, Ø43H, ØABH, Ø3FH);

 85    1           DCL CHECK BYTE DATA (ØØ2H),      /***  GT  **/
                       CHECK1 BYTE DATA (ØØ3H),     /***  GE  **/
                       CHECK2 BYTE DATA (ØØ1H),     /***  LE  **/
                       CHECK3 BYTE DATA (ØØ4H);     /***  EQ  **/


                 /**************************************************************
                 -   ****
                     *
                     *   BINARY TO FLOATING POINT FORMAT
```

```
                   *
                   *  THIS PROCEDURE IS USED TO CONVERT A NUMBER FROM INTERNAL(1
               -   INARY
                   *  -FROM ADC/DAC BOARD) FORM TO AN INTEL FLOATING POINT FORM
               -   AT.
                   *
                   *  PARAMETERS:
                   *
                   *     BIN - BINARY VALUE IN MEMORY THAT CONNECTED TO ADC/DAC B
               -   OARD.
                   *     FLT - ARRAY THAT FLOATING POINT FORMAT LOCATED.
                   *
                   ******************************************************************
               -   ****/
  86   1           BINARY$TO$FLOAT: PROCEDURE(BIN,FLAD) PUBLIC;
  87   2             DCL FLAD ADDRESS;
  88   2             DCL (BIN, MNT, EXP, ISAR, COUNT) BYTE;
  89   2             DCL (FLT BASED FLAD)(4) BYTE;
                   /*** TO DETERMINE THE BINARY NUMBER IS POSITIVE OR NEGATIVE I
               -   F IT
                        IF NEGATIVE TO TAKE 2'S COMPLEMENT.  ***/
  90   2             IF (BIN AND 80H) <> 0 THEN
  91   2               DO;
  92   3                 MNT = NOT(BIN);
  93   3                 MNT = ( MNT + 1);
  94   3                 ISAR = 1;
  95   3               END;
                     ELSE
  96   2               DO;
  97   3                 MNT = BIN;
  98    3              ISAR = 0;
  99    3            END;
 100    2          COUNT = 0;
        /***   TO FIND THE MANTISSA    ***/
 101    2          DO WHILE (MNT AND 80H) = 0;
 102    3            MNT = SHL(MNT, 1);
 103    3            COUNT = (COUNT + 1);
 104    3          END;
 105    2          MNT = SHL(MNT, 1);
 106    2          COUNT = (7 - COUNT);
 107    2          MNT = SHR(MNT, 1);
 108    2          EXP = (7FH + COUNT);
 109    2          IF (EXP AND 01H) <> 0 THEN
 110    2            MNT = (MNT OR 80H);
                   ELSE
 111    2            MNT = (MNT AND 7FH);
        /***   TO DETERMINE THE SIGN BIT    ***/
 112    2          EXP = SHR(EXP, 1);
 113    2          IF ISAR = 1 THEN
 114    2            EXP = (EXP OR 80H);
                   ELSE
 115    2            EXP = (EXP AND 7FH);
        /***   TO LOCATE THE BINARY NUMBER TO FLOATING POINT FORMAT
      -    ***/
 116    2          FLT(0) = 00H;
 117    2          FLT(1) = 00H;
 118    2          FLT(2) = MNT;
 119    2          FLT(3) = EXP;
 120    2        END BINARY$TO$FLOAT;
```

```
      /*************************************************************************
  -    ****
      *
      .   BINARYSTOASCII:
      *
      *   THIS PROCEDURE IS USED TO CONVERT A NUMBER FROM INTERNAL
      *   (BINARY) FORM TO AN ASCII STRING SUITABLE FOR PRINTING.
      *
      *   PARAMETERS:
      *
      *          VALUE    - THE NUMBER WHOSE PRINTED REPRESENTATION I
  -    S
                            DESIRED.
      *
      *          BASE     - AN INTEGER BETWEEN 2 AND 16, INCLUSIVE, S
  -    PE-                  CIFYING IN WHAT NUMBER BASE 'VALUE' IS TO
      *
  -     BE                  INTERPRETED.
      *
      *
      *          LC       - (LEADING CHARECTER): LEADING ZEROS IN TH
  -                         PRINTED REPRESENTATION WILL BE DESIGNATED
      *
  -     BY                  'LC', WHICH SHOULD BE AN ASCII CHARECTER.
      *
  -     U-                  SEFUL VALUES OF 'LC', WHICH SHOULD BE AN
      *
  -    ASCII                CHARECTER. USEFUL VALUES OF 'LC' ARE: 0(N
      *
  -    ULL),                ' '(SPACE), AND '0'(ZERO).
      *
```

```
                *        BUFADR    - THE ADDRESS OF A BUFFER OF AT LEAST 'WIDT
           -     H' BYTES
                *                    INTO WHICH THE PRINTED REPRESENTATION IS
           -     PLACED.
                *        WIDTH     - THE NUMBER OF CHARACTER POSITIONS DESIRED
           -      IN THE
                *                    PRINTED REPRESENTATION.
                *
                *                    BASE > 1 AND WIDTH >0
                *
                ********************************************************************
           -     *****/
 121    1       BINARY$TO$ASCII: PROCEDURE(VALUE, BASE, LC, BUFADR, WIDTH) PU
           -     BLIC;
 122    2         DCL (BUFADR) ADDRESS;
 123    2         DCL (BASE, LC, WIDTH, I, VALUE) BYTE;
 124    2         DCL (CHARS BASED BUFADR) (1) BYTE;
 125    2         DCL DIGITS (*) BYTE DATA('0123456789ABCDEF');
 126    2         DO I = 1 TO WIDTH;
 127    3               CHARS(I-1) = DIGITS(VALUE MOD BASE);
 128    3               VALUE = VALUE / BASE;
 129    3         END;
 130    2         I = 0;
 131    2         DO WHILE CHARS(I) = '0' AND I < WIDTH - 1;
 132    3               CHARS(I) = ' ';
 133    3               I = I + 1;
 134    3         END;
 135    2       END BINARY$TO$ASCII;

                /*******************************************************************
```

```
                  *
                  *    FLOATING POINT FORMAT TO BINARY
                  *
                  *    THIS PROCEDURE IS USED TO CONVERT A NUMBER FROM INTERNAL
                  *    FLOATING POINT FORMAT TO INTERNAL BINARY.
                  *
                  *    PARAMETERS:
                  *
                  *    RBIN - BINARY VALUE IN THE MEMORY ASSOCIATED TO ADC/DAC
                  *    FLT  - ARRAY THAT FLOATING POINT FORMAT LOCATED
                  *
                  *
                  ***********************************************************************/
 136   1          FLOAT$TO$BINARY: PROCEDURE(RFLAD, RBIN) PUBLIC;
 137   2             DECLARE RFLAD ADDRESS;
 138   2             DCL (RBIN, MNT, ISAR, EXP, COUNT) BYTE;
 139   2             DCL (FLT BASED RFLAD) (4) BYTE;
 140   2             MNT = FLT(2);
 141   2             EXP = FLT(3);
 142   2             IF (EXP AND 80H) <> 0 THEN
 143   2                ISAR = 1;
                       ELSE
 144   2                ISAR = 0;
 145   2             EXP = SHL(EXP, 1);
 146   2             IF (MNT AND 80H) <> 0 THEN
 147   2             EXP = EXP +1;
 148   2             MNT = SHL(MNT,1);
 149   2             MNT = SHR(MNT,1);
 150   2             MNT = (MNT OR 80H);
 151   2             COUNT = EXP - 7FH;
```

```
152   2         COUNT = 7 - COUNT;
153   2         MNT = SHR(MNT, COUNT);
154   2         IF (ISAR = 1) THEN
155   2           DO;
156   3              MNT = NOT(MNT);
157   3              MNT = MNT + 1;
158   3           END;
159   2         RBIN = MNT;
160   2       END FLOATSTOSBINARY;

              /*****************************************************************
        -      ****
              *
              *   SCALE:
              *
              *   THIS PROCEDURE IS USED TO SCALE INPUT CHANNEL DATAS TO TH
        -      E
              *   REAL DATA SCALES AS FOLLOWING:
              *
              *   RANGE = 99999YRD
              *   ELEVATION = 0 TO 90 DEGREES
              *   BEARING   = 0 TO 359 DEGREES.
              *   VX        = +- 500 YRD/SEC
              *   VY        = +- 500 YRD/SEC
              *   VZ        = +- 170 YRD/SEC
              *
              *
              ******************************************************************
        -      */
161   1       SCALE: PROCEDURE;
```

```
162   2          CALL BINARY$TO$FLOAT(BIN$RNG, .FRNG(0));
163   2          CALL BINARY$TO$FLOAT(BIN$ELV, .FELV(0));
164   2          CALL BINARY$TO$FLOAT(BIN$BRG, .FBRG(0));
165   2          CALL BINARY$TO$FLOAT(BIN$VX, .FVX(0));
166   2          CALL BINARY$TO$FLOAT(BIN$VY, .FVY(0));
167   2          CALL BINARY$TO$FLOAT(BIN$VZ, .FVZ(0));

           /*** SCALE RANGE   FRNG = FRNG * (99999 / 127)
                 "    BEARING   FBRG = FBRG * (360 /127)
                      ELEVATION FELV = FELV * (90 / 127)     ***/

168   2          CALL FMUL(.FRNG, .FP$787$39, .FRNG);
169   2          CALL FMUL(.FBRG, .FP$2$83, .FBRG);
170   2          CALL FMUL(.FELV, .FP$0$708, .FELV);

           /***  SCALE SPEED COMPONENTS  ***/

171   2          CALL FMUL(.FVX, .FP$3$93, .FVX);
172   2          CALL FMUL(.FVY, .FP$3$93, .FVY);
173   2          CALL FMUL(.FVZ, .FP$1$33, .FVZ);

           /***   CONVERPTION OF BRG. AND ELV. TO RADIAN   ***/

174   2          CALL FMUL(.FELV, .DEG$TO$RAD, .FELV);
175   2          CALL FMUL(.FBRG, .DEG$TO$RAD, .FBRG);

176   2       END SCALE;

           /*****************************************************************
```

```
              -   **
                  *
                  *   GENSTRGSVALI:
                  *
                  *     THIS PROCEDURE IS USED TO CALCULATE THE FOLLOWING PARA-
                  *     METERS.
                  *
                  *
                  *     PARAMETERS:
                  *
                  *       CT     - TARGET COURSE
                  *       SH     - AIR TARGET HORIZANTAL GROUND SPEED
                  *       SR     - RELATIVE TARGET SPEED
                  *       TBR    - TRUE TARGET BEARING
                  *       TSANG  - TARGET ANGLE
                  *
                  *****************************************************************
              -    **/
 177    1         GENSTRGSVALI:PROCEDURE;
 178    2          DCL CHECK BYTE DATA (002H),     /*** GREATERR THAN    ***/
                        CHECK1 BYTE DATA (003H),    /***    GE      ***/
                        CHECK2 BYTE DATA (001H),    /***    LE      ***/
                        CHECK3 BYTE DATA (004H);    /***    EQ      ***/

 179    2          DCL PI$FLOAT (4) BYTE DATA(0DBH, 0FH, 49H, 40H),   /** 3.1
              -    41593  **/
                        TWO$PI (4) BYTE DATA (0DBH, 0FH, 0C9H, 40H),   /** 6.2
              -    831853 **/
                        PI$OV$TWO (4) BYTE DATA(0DBH, 0FH, 0C9H, 3FH),
                        TRE$PI$OV$2 (4) BYTE DATA(0E4H, 0CBH, 96H, 40H);
```

```
180   2         DCL TMP$1 (4) BYTE,
                    TMP$2 (4) BYTE,
                    TMP$3 (4) BYTE;

               /*** CALCULATION OF SH AND SR
                    SH = SQRT( VX**2 + VY**2 )
                    SR = SQRT( VZ**2 + SH**2 )        ***/

181   2         CALL FSQR(.FVX, .TMP$1);
182   2         CALL FSQR(.FVY, .TMP$2);
183   2         CALL FADD(.TMP$1, .TMP$2, .FSH);
184   2         CALL FSQR(.FVZ, .TMP$3);
185   2         CALL FADD(.TMP$3, .FSH, .FSR);
186   2         CALL FSQRT(.FSH, .FSH);
187   2         CALL FSQRT(.FSR, .FSR);

               /***   TO FIND TARGET COURSE   ***/

188   2         CALL ARCSTAN(.FVX, .FVY, .FCT);
189   2         IF FZTST(.FVX, .CHECK) THEN                         /* IF FVX >
          -     0 */
190   2            DO;
191   3               IF FZTST(.FVY, .CHECK) THEN       /* IF FVY > 0 , QUAD=1
          -     */
192   3                  CALL FSUB(.PI$OVSTWO, .FCT, .FCT);
                      ELSE
193   3                  DO;                             /* FVY < 0 , QUAD = 4
          -     */
```

```
194   4                 CALL FSUB(.TWO$PI, .FCT, .FCT);
195   4                 CALL FADD(.PI$OV$TWO, .FCT, .FCT);
196   4               END;
197   3           END;
               ELSE                                          /*     FVY < 2
             /
198   2           DO;
199   3             IF FZTST(.FVY, .CHECK) THEN
200   3               DO;
201   4                CALL FSUB(.FCT, .PI$OV$TWO, .FCT);
202   4                CALL FSUB(.TWO$PI, .FCT, .FCT);
203   4               END;
                    ELSE
204   3               DO;
205   4                CALL FSUB(.FCT, .PI$OV$TWO, .FCT);
206   4                CALL FSUB(.TWO$PI, .FCT, .FCT);
207   4               END;
208   3             END;
            /***   GET OWN COURSE FROM OTHER MODULE FOR NOW ASSUME CONSTA
             NT  ***/

            /***   TRUE TARGET BEARING     B = BR + COWN      ***/

229   2        CALL FADD(.FBRG, .FCOWN, .FB$TRU);
210   2        IF FCMPR(.FB$TRU, .TWO$PI, .CHECK1) THEN
211   2           CALL FSUB(.FB$TRU, .TWO$PI, .FB$TRU);

            /***   TARGET ANGLE   A = B + 180 - CT     ***/

212   2        CALL FADD(.FB$TRU, .PI$FLOAT, .FT$ANG);
```

```
213   2          CALL FSUB(.FT$ANG, .FCT, .FT$ANG);
214   2          IF FCMPR(.FT$ANG, .TWO$PI, .CHECK1) THEN
215   2            CALL FSUB(.FT$ANG, .TWO$PI, .FT$ANG);

216   2        END GEN$TRG$VALI;

               /*************************************************************
          -     ****
               *
               *   GEN$TRG$VAL$II:
               *
               *   THIS PROCEDURE IS USED TO CALCULATE THE FOLLOWING PARAMET
          -     ERS.
               *
               *   FXO - CROSS COMPONENT OF OWN-SHIP VELOCITY
               *   FYO - LINE      "       "   "     "
               *   FYT -   "       "       "   TARGET    "
               *   FXT - CROSS     "       "   OWN-SHIP
               *
               *************************************************************
          -     ****/
217   1        GEN$TRG$VAL$II: PROCEDURE;
218   2          CALL COS$SIN(.FBRG, .COS$FBRG, .SIN$FBRG);
219   2          CALL FMUL(.COS$FBRG, .FSO, .FYO);
220   2          CALL FMUL(.SIN$FBRG, .FSO, .FXO);
221   2          CALL COS$SIN(.FT$ANG, .COS$FT$ANG, .SIN$FT$ANG);
222   2          CALL FMUL(.COS$FT$ANG,.FSH, .FYT);
223   2          CALL FMUL(.SIN$FT$ANG, .FSH, .FXT);
224   2        END GEN$TRG$VAL$II;
```

```
         /*********************************************************************
       -  *****
          *
          *   GENSTRGSVAL$III:
          *
          *    THIS PROCEDURE IS USED TO CALCULATE THE FOLLOWING PARAME
       -  TERS
          *
          *    PARAMETERS:
          *    FDRH - HORIZONTAL RANGE RATE
          *    FRDBS - LINEAR DEFLECTION RATE
          *
          *    FDR - RANGE RATE
          *    FDE - ANGULAR ELEVATION RATE
          *    FDBR - RELATIVE ANGULAR BEARING RATE
          *
          *    DRH = -(YO + YT)
          *    RDBS = XO + XT
          *    DR = VZ * SIN(E) + DRH * COS(E)
          *    RDE = DH * COS(E) - DRH * SIN(E)
          *    DEE = RDBS / (R * COS(E))
          *
          *
         **********************************************************************
       -  *****/
225   1   GENSTRGSVAL$III: PROCEDURE;
226   2   DCL FPSMINS1 (4) BYTE DATA (00H, 00H, 80H, 0FFH);
227   2   DCL COSSFELV (4) BYTE,
              SINSFELV (4) BYTE,
                      FVZ$SINE (4) BYTE,
                      FDRH$COSE (4) BYTE,
                      FDRH$SINE (4) BYTE,
                      FR$COSE (4) BYTE,
                      FVZ$COSE (4) BYTE;

                    /***     FDRE = - ( FYO + FYT )    ***/
 228    2           CALL FADD(.FYO, .FYT, .FDRE);
 229    2           CALL FMUL(.FDRE, .FP$MINS1, .FDRE);

                    /***   FRDES = FXO + FXT      ***/

 230    2           CALL FADD(.FXO, .FXT, .FRDES);
                    /***   FDR = FVZ * SIN(ELV) + FDRH * COS(FELV)    ***/

 231    2           CALL COS$SIN(.FELV, .COS$FELV, .SIN$FELV);
 232    2           CALL FMUL(.FVZ, .SIN$FELV, .FVZ$SINE);
 233    2           CALL FMUL(.FDRH, .COS$FELV, .FDRH$COSE);
 234    2           CALL FADD(.FVZ$SINE, .FDRH$COSE, .FDE);
                    /***   FRDE = FDH*COS(FELV) - FDRH*SIN(FELV)    ***/

 235    2           CALL FMUL(.FVZ, .COS$FELV, .FVZ$COSE);
 236    2           CALL FMUL(.FDRE, .SIN$FELV, .FDRH$SINE);
 237    2           CALL FSUB(.FVZ$COSE, .FDRH$SINE, .FRDE);
                    /***     FDBR = FRDES / ( FRNG * COS(FELV)    ***/

 238    2           CALL FMUL(.FRNG, .COS$FELV, .FR$COSE);
 239    2           CALL FDIV(.FRDES, .FR$COSE, .FDBE);
                    /***     FDE = HDE / R    ***/
```

```
240    2       CALL FDIV(.FRDE, .FRNG, .FDE);
241    2       END GENSTRGSVALSIII;
               /*************************************************************
          -     **
               *
               *   GENSTRGSVALSIV:
               *
                     THIS PROCEDURE IS USED TO CALCULATE THE FOLLOWING PARAM
          -     ETERS
               *
               *    PARAMETERS:
               *
               *    CR   - GENERATED PRESENT RANGE
               *    CBR  - GENERATED RELATIVE TARGET BEARING
               *    CE   -     "      TARGET ELEVATION
               *    H    -     "      PRESENT ALTITUDE
               *    DT   - TIME INCREMENT
               *
               **************************************************************
          -     **/
242    1       GENSTRGSVALSIV: PROCEDURE;
243    2       DCL FDTSDR (4) BYTE,
                   FDTSDE (4) BYTE,
                   FDTSDBR (4) BYTE;
244    2       DCL FDT (4) BYTE DATA (00H, 00H, 80H, 3FH);

               /*** FCR = FRNG + FDT*FDR    ***/

245    2       CALL FMUL(.FDT, .FDR, .FDTSDR);
```

```
246   2      CALL FADD(.FRNG, .FDT$DR, .FCR);

             /***   CER = ERG + DT*DBR     ***/

247   2      CALL FMUL(.FDT, .FDER, .FDT$DER);
248   2      CALL FADD(.FERG, .FDT$DER, .FCER);

             /***   CE = ELV + DT*DE       ***/

249   2      CALL FMUL(.FDT, .FDE, .FDT$DE);
250   2      CALL FADD(.FELV, .FDT$DE, .FCE);

             /***   H = CR*SIN(CE)    ***/

251   2      CALL COS$SIN(.FCE, .COS$FCE, .SIN$FCE);
252   2      CALL FMUL(.FCR, .SIN$FCE, .FH);

253   2      END GEN$TPG$VAL$IV;

             /*****************************************************************
             *
             *   TOSGET$TF:
             *
             *     THIS PROCEDURE IS USED TO GET TIME OF FLIGHT
             *
             *****************************************************************/

254   1      TOSGET$TF: PROCEDURE;
255   2      DCL INSVEL (4) BYTE DATA (0AVH, 0AAH, 58H, 44H);    /**   SEC.
           -  66 YRD/SEC */
256   2      DCL COS$FELV (4) BYTE,
                    INS$VELSX (4) BYTE,
                    FCRX (4) BYTE,
                    SINSFELV (4) BYTE;
 257   2        CALL COS$SIN(.FELV, .COS$FELV, .SINSFELV);
 258   2        CALL FMUL(.INS$VEL, .COS$FELV, .INS$VELSX);
 259   2        CALL FMUL(.FCR, .COS$FELV, .FCRX);
 260   2        CALL FDIV(.FCRX, .INS$VELSX, .TF);
 261   2        END TOSGETSTF;

                /*************************************************************
                 *
                 *
                 *  PREDICTED$TRG$VAL:
                 *
                 *   THIS PROCEDURE IS USED TO CALCULATE THE PREDICTED TARGET
                 *   RANGE, BEARING, AND ELEVATION.
                 *
                 *   PARAMETERS:
                 *
                 *   PRNG  - PREDICTED TARGET RANGE
                 *   PBRG  -     "        "   BEARING
                 *   PELV  -     "        "   ELEVATION
                 *
                 *************************************************************
                 */
 262   1        PRED$TRG$VAL: PROCEDURE;
 263   2        DCL TEMP1 (4) BYTE,
                    TEMP2 (4) BYTE,
                    TEMP3 (4) BYTE,
                    TEMP4 (4) BYTE,
                    TEMP5 (4) BYTE,
                    TEMP6 (4) BYTE,
                    TEMP7 (4) BYTE,
                    TEMP8 (4) BYTE,
                    TEMP9 (4) BYTE;

 264   2        DCL FVT (4) BYTE,
                    COS$PELV (4) BYTE,
                    SIN$PELV (4) BYTE,
                    R2$COSE2 (4) BYTE,
                    FDTH (4) BYTE;

                /***    PREDICTED RANGE  ***/
 265   2        CALL FMUL(.TF, .FRDE, .TEMP1);
 266   2        CALL FSQR(.TEMP1, .TEMP4);
 267   2        CALL FMUL(.TF, .FRTES, .TEMP2);
 268   2        CALL FSQR(.TEMP2, .TEMP5);
 269   2        CALL FMUL(.TF, .FDR, .TEMP3);
 270   2        CALL FADD(.FRNG, .TEMP3, .TEMP$6);
 271   2        CALL FSQR(.TEMP6, .TEMP6);
 272   2        CALL FADD(.TEMP4, .TEMP5, .TEMP8);
 273   2        CALL FADD(.TEMP8, .TEMP5, .TEMP9);
 274   2        CALL FSQRT(.TEMP9, .PRNG);

                /***    PREDICTED ELEVATION   ***/
 275   2        CALL FDIV(.TEMP1, .PRNG, .FVT);
 276   2        CALL FADD(.FVT, .FELV, .PELV);

                /***   PREDICTED DEFLECTION   ***/
 277   2        CALL COS$SIN(.PELV, .COS$PELV, .SIN$PELV);
```

```
278   2      CALL FMUL(.PRNG, .COS$PELV, .R2$COSE2);
279   2      CALL FDIV(.TEMP2, .R2$COSE2, .FDTH);
280   2      CALL FADD(.FBRG, .FDTH, .PBRG);

             /***    PREDICTED ALTITUTE   ***/
281   2      CALL FMUL(.PRNG, .SIN$PELV, .PH);

282   2      END PREDSTRGSVAL;

             /*****************************************************
             *
             *   INVERSE SCALE:
             *
             *   THIS PROCEDURE IS USED TO DTERMINE LIMITS AND
             *   CONVERT TO DEGREES. ALSO IT HELP LINIERAZATIONS.
             *
             *****************************************************/
283   1      INV$SCALE: PROCEDURE;

284   2      DCL FP$1$29 (4) BYTE DATA (0EEH, 0A2EH, 0EBH, 03FH);
             /***   RADIAN TO DEGREE   ***/

285   2      CALL FDIV (.PBRG, .DEG$TO$RAD, .OBRG);
286   2      CALL FDIV (.PELV, .DEG$TO$RAD, .OELV);

             /***   TO DETERMINE PORT AND STARBOARD LIMITS   ***/

287   2      IF FCMPR(.OBRG, .FP$352, .CHECK1) THEN
288   2         CALL FSUB(.OBRG, .FP$360, .OBRG);
```

```
289   2      IF FCMPR (.OBRG, .FP$180, .CHECK1) THEN
290   2        DO;
291   3          IF FCMPR (.OBRG, .FP$225, .CHECK2) THEN
292   3            DO;
293   4           OBRG(0) = FP$NG$135(0);
294   4               OBRG(1) = FP$NG$135(1);
295   4               OBRG(2) = FP$NG$135(2);
296   4               OBRG(3) = FP$NG$135(3);
297   4            END;
             ELSE
298   3            DO;
299   4               CALL FSUB(.FP$360, .OBRG, .OBRG);
300   4               CALL FMUL(.OBRG, .MIN$1, .OBRG);
301   4            END;
302   3        END;
             ELSE
303   2        DO;
304   3          IF FCMPR(.OBRG, .FP$135, .CHECK1) THEN
305   3            DO;
306   4               OBRG(0) = FP$135(0);
307   4               OBRG(1) = FP$135(1);
308   4               OBRG(2) = FP$135(2);
309   4               OBRG(3) = FP$135(3);
310   4            END;
311   3        END;

             /***   SYSTEM LINEAR OUTPUT SCALE  ***/

312   2      CALL FMUL(.OBRG, .FP$1$09, .OBRG);
313   2      CALL FMUL(.OELV, .FP$1$09, .OELV);
```

```
 314   2       END INVSSCALE;

               /*************************************************
               *
               *   ITERATION:
               *
               *   THIS PROCEDURE IS USED TO GENERATE RELATIVE
               *   TARGET POSITION FOREVER.
               *
               *************************************************/
 315   1       ITERATION: PROCEDURE;
 316   2       FRNG(0) = FCR(0);
 317   2       FRNG(1) = FCR(1);
 318   2       FRNG(2) = FCR(2);
 319   2       FRNG(3) = FCR(3);

 320   2       FBRG(0) = FCBR(0);
 321   2       FBRG(1) = FCBR(1);
 322   2       FBRG(2) = FCBR(2);
 323   2       FBRG(3) = FCBR(3);

 324   2       FELV(0) = FCE(0);
 325   2       FELV(1) = FCE(1);
 326   2       FELV(2) = FCE(2);
 327   2       FELV(3) = FCE(3);

 328   2       END ITERATION;

 329   1       CALL INITSFP;
```

```
330   1      CALL SCALE;

331   1      CALL GEN$TRG$VALI;

332   1      CALL GEN$TRG$VAL$II;

333   1      CALL GEN$TRG$VAL$III;

334   1      CALL GEN$TRG$VAL$IV;

335   1      CALL TO$GET$TF;

336   1      CALL PRED$TRG$VAL;

337   1      DO WHILE 1;          /*** DO FOREVER     ***/
338   2         CALL GEN$TRG$VALI;
339   2         CALL GEN$TRG$VALII;
340   2         CALL GEN$TRG$VALIII;
341   2         CALL GEN$TRG$VALIV;
342   2         CALL TO$GET$TF;
343   2         CALL PRED$TRG$VAL;
344   2         CALL INV$SCALE;
345   2         CALL ITERATION;
346   2         CALL FLOAT$TO$BINARY(.OBRG(0), OBNERG);
347   2         CALL FLOAT$TO$BINARY(.OELV(0), OBNELV);

348   2      END;
```

```
PL/M-80 COMPILER                                              20 MAR 81  PAGE 2-

 349   1        CALL EXIT;

 350   1        END AAW$GFCS;


MODULE INFORMATION:

      CODE AREA SIZE      = 08E6H   2054D
      VARIABLE AREA SIZE  = 01ACH    428D
      MAXIMUM STACK SIZE  = 0006H      6D
      759 LINES READ
      0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER                                       20 MAR 81   PAGE   1

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SWGFCS
OBJECT MODULE PLACED IN SW.OBJ
COMPILER INVOKED BY:  :F1:PLM80 SW.SRC PAGELENGTH(33) PAGEWIDTH(75) DATE(20
                    - MAR 81)


   1            SWGFCS: DO;

                /***   EXTERNALS   ***/

   2    1       CRT$WRITE:
                  PROCEDURE (CHAR) EXTERNAL;
   3    2         DECLARE CHAR BYTE; END;

   5    1       CRT$PRINT$STRING:
                  PROCEDURE (A) EXTERNAL;
   6    2         DECLARE A ADDRESS; END;

   8    1       SEND$CRLF:
                  PROCEDURE EXTERNAL;
   9    2         END;

  10    1       SEND$SPACE:
                  PROCEDURE (NUM) EXTERNAL;
  11    2         DECLARE NUM BYTE; END;

  13    1       FADD:
                  PROCEDURE (A,B,C) EXTERNAL;
  14    2         DECLARE (A,B,C) ADDRESS; END;
```

```
 15   1        FSUB:
                 PROCEDURE (A,B,C) EXTERNAL;
 17   2          DECLARE (A,B,C) ADDRESS; END;
 19   1        FMUL:
                 PROCEDURE (A,B,C) EXTERNAL;
 20   2          DECLARE (A,B,C) ADDRESS; END;
 22   1        FDIV:
                 PROCEDURE (A,B,C) EXTERNAL;
 23   2          DECLARE (A,B,C) ADDRESS; END;
 25   1        FSQR:
                 PROCEDURE (A,B) EXTERNAL;
 26   2          DECLARE (A,B) ADDRESS; END;
 28   1        FSORT:
                 PROCEDURE (A,B) EXTERNAL;
 29   2          DECLARE (A,B) ADDRESS; END;
 31   1        FCMPR:
                 PROCEDURE (A,B,C) BYTE EXTERNAL;
 32   2          DECLARE (A,B,C) ADDRESS; END;
 34   1        FZTST:
                 PROCEDURE (A,B) BYTE EXTERNAL;
 35   2          DECLARE (A,B) ADDRESS; END;
 37   1        EXCH:
                 PROCEDURE (A,B) EXTERNAL;
 38   2          DECLARE (A,B) ADDRESS; END;
 40   1        COS$SIN:
                 PROCEDURE (A,B,C) EXTERNAL;
 41   2          DECLARE (A,B,C) ADDRESS; END;
 43   1        ARCSTAN:
                 PROCEDURE (A,B,C) EXTERNAL;
 44   2          DECLARE (A,B,C) ADDRESS; END;
```

```
  46    1       FLOAT$TO$ASCII:
                  PROCEDURE (A,B,C) EXTERNAL;
  47    2         DECLARE (A,B,C) ADDRESS; END;
  49    1       INIT$FP:
                  PROCEDURE EXTERNAL;
  50    2         END;
  51    1       FP$FORMAT:
                  PROCEDURE (A, B, C, D) BYTE EXTERNAL;
  52    2         DECLARE (A,B) ADDRESS, (C,D) BYTE; END;

  54    1       WRITE:
                  PROCEDURE (AFT,BUFFER,LENGTH,STATUS) EXTERNAL;
  55    2         DECLARE (AFT,BUFFER,LENGTH,STATUS) ADDRESS;
  56    2         END WRITE;
  57    1       EXIT:
                  PROCEDURE EXTERNAL;
  58    2         END EXIT;

                /****   DECLERATIONS   ***/

  59    1       DECLARE LIT LITERALLY 'LITERALLY',
                           DCL LIT 'DECLARE';
  60    1       DCL BUFFER (128) BYTE;
  61    1       DCL STATUS ADDRESS;
  62    1       DCL CRLF(2) BYTE DATA (0DH, 0AH);
  63    1       DCL PINSBRG BYTE AT (0F700H),
                    PINSELV BYTE AT (0F701H);

  64    1       DCL FRNG (4) BYTE,
                    FELV (4) BYTE,
                     FBRG (4) BYTE;

 65    1       DCL FXO (4) BYTE,
                   FXT (4) BYTE,
                   FYO (4) BYTE,
                   FYT (4) BYTE;

 66    1       DCL FSA (4) BYTE,
                   FB (4) BYTE,
                   COS$FBRG (4) BYTE,
                   SIN$FBRG (4) BYTE,
                   COS$FSA (4) BYTE,
                   SIN$FSA (4) BYTE;

 67    1       DCL FDR (4) BYTE,
                   FRDPR (4) BYTE,
                   FDBR (4) BYTE;

 68    1       DCL FSO (4) BYTE DATA (00H, 00H, 80H, 3FH),
                   FST(4) BYTE DATA (00H, 00H, 0F0H, 41H),
                   FCT (4) BYTE DATA (00H, 00H, 96H, 42H),
                   FRER (4) BYTE DATA (00H, 00H, 96H, 43H),
                   FGRNG (4) BYTE DATA (00H, 60H, 6AH, 46H),
                   FGELV (4) BYTE DATA (00H, 00H, 0ECH, 41H),
                   FCO (4) BYTE DATA (00H, 80H, 0A2H, 43H);

 69    1       DCL TF (4) BYTE DATA (29H, 5CH, 21H, 42H);

 70    1       DCL FCR (4) BYTE,
                   FCBR (4) BYTE;
```

```
  [illegible]   1        DCL TE[illegible] (4) BYTE,
                             [illegible] (4) BYTE,
                             TE[illegible] (4) BYTE;

  [illegible]   1        DCL VAL[illegible] [illegible];

  [illegible]   1        DCL PLUS [illegible] LIT '[illegible]',
                             MINUS [illegible] LIT '[illegible]',
                             COLON LIT '[illegible]',
                             POINT LIT '[illegible]',
                             BLANK LIT '[illegible]';

  [illegible]   1        DCL DECSTR[illegible] (4) BYTE DATA([illegible], [illegible], [illegible], [illegible]
                          [illegible] */
  [illegible]   1        DCL [illegible] (4) BYTE DATA ([illegible], [illegible], [illegible], [illegible]);

                         /*********************************************************************
                          ****
                          *
                          *    BINARYTOASCII:
                          *
                          *    THIS PROCEDURE IS USED TO [illegible] A NUMBER [illegible]
                          *    [illegible] FORM TO AN ASCII [illegible]
                          *
                          *    PARAMETERS:
                          *
                          *          [illegible]       - THE NUMBER [illegible]
```

566

```
      *                                  DESIRED.
      *           BASE       - AN INTEGER BETWEEN 2 AND 16, INCLUSIVE, S
-      PE-
      *                                  CIFYING IN WHAT NUMBER BASE 'VALUE' IS TO
-       BE
      *                                  INTERPRETED.
      *
      *           IC         - (LEADING CHARACTER): LEADING ZEROS IN THE
-
      *                                  PRINTED REPRESENTATION WILL BE DESIGNATED
-       BY
      *                                  'LC', WHICH SHOULD BE AN ASCII CHARACTER.
-       U-
      *                                  SEFUL VALUES OF 'LC', WHICH SHOULD BE AN
-      ASCII
      *                                  CHARECTER. USEFUL VALUES OF 'LC' ARE: 0(N
-      ULL),
      *                                  ' '(SPACE), AND '0'(ZERO).
      *           BUFADR     - THE ADDRESS OF A BUFFER OF AT LEAST 'WIDT
-      H' BYTES
      *                                  INTO WHICH THE PRINTED REPRESENTATION IS
-      PLACED.
      *           WIDTH      - THE NUMBER OF CHARECTER POSITIONS DESIRED
-       IN THE
      *                                  PRINTED REPRESENTATION.
      *
      *                                  BASE > 1 AND WIDTH >0
      *
      ****************************************************************
-      ****/
```

```
76   1      BINARY$TO$ASCII: PROCEDURE(VALUE, BASE, LC, BUFADR, WIDTH) PU
          -  BLIC;
77   2        DCL (BUFADR) ADDRESS;
78   2        DCL (BASE, LC, WIDTH, I, VALUE) BYTE;
79   2        DCL (CHARS BASED BUFADR) (1) BYTE;
80   2        DCL DIGITS (*) BYTE DATA('0123456789ABCDEF');
81   2        DO I = 1 TO WIDTH;
82   3              CHARS(I-1) = DIGITS(VALUE MOD BASE);
83   3              VALUE = VALUE / BASE;
84   3        END;
85   2        I = 0;
86   2        DO WHILE CHARS(I) = '0' AND I < WIDTH - 1;
87   3              CHARS(I) = ' ';
88   3              I = I + 1;
89   3        END;
90   2      END BINARY$TO$ASCII;

            /*****************************************************
            *
            *   FLOATING POINT FORMAT TO BINARY:
            *
            *   THIS PROCEDURE IS USED TO CONVERT A NUMBER FROM
            *   FLOATING POINT FORMAT TO BINARY.
            *
            *   PARAMETERS:
            *
            *   PBIN - BINARY VALUE IN MEMORY THAT CONNECTED TO ADC/DAC
            *   FLT - ARRAY THAT FLOATING POINT FORMAT LOCATED
            *
            *****************************************************/
```

```
 91   1      FLOAT$TO$BINARY: PROCEDURE(RFLAD, RBIN);
 92   2         DCL RFLAD ADDRESS;
 93   2         DCL (RBIN, MNT, ISAR, EXP, COUNT) BYTE;
 94   2         DCL (FLT BASED RFLAD) (4) BYTE;

 95   2         MNT = FLT(2);
 96   2         EXP = FLT(3);

 97   2         IF (EXP AND 80H) <> 0 THEN
 98   2            ISAR = 1;
                ELSE
 99   2            ISAR = 0;
100   2         EXP = SHL(EXP, 1);

101   2         IF (MNT AND 80H) <> 0 THEN
102   2           EXP = EXP + 1;
103   2         MNT = SHL(MNT, 1);
104   2         MNT = SHR(MNT, 1);
105   2         MNT = (MNT OR 80H);
106   2         COUNT = EXP - 7FH;
107   2         COUNT = 7 - COUNT;
108   2         MNT = SHR(MNT, COUNT);

109   2         IF (ISAR = 1) THEN
110   2            DO;
111   3              MNT = NOT(MNT);
112   3              MNT = MNT +1;
113   3            END;
114   2         RBIN = MNT;
115   2      END FLOAT$TO$BINARY;
```

```
              /*****************************************************************
            -  **
               *
               *   SW$GEN$TRG$VALI:
               *
               *     THIS PROCEDURE IS USED TO CALCULATE THE FOLLOWING PARA-
               *     METERS.
               *
               *
               *     PARAMETERS:
               *
               *       B   - TRUE TARGET BEARING
               *       A - TARGET ANGLE
               *
               *****************************************************************
            -  **/
 116    1     SW$GEN$TRG$VALI: PROCEDURE;
 117    2       DCL CHECK BYTE DATA (002H),    /*** GREATHER THAN    ***/
                    CHECK1 BYTE DATA (003H),   /***    GE     ***/
                    CHECK2 BYTE DATA (001H),   /***    LE     ***/
                    CHECK3 BYTE DATA (004H);   /***    EQ     ***/

 118    2       DCL PISFLOAT (4) BYTE DATA(0DBH, 0FH, 49H, 40H),   /**  3.1
            -  41593  **/
                    TWOSPI (4) BYTE DATA (2DBH, 0FH, 0C9H, 40H),   /**  6.2
            -  831853 **/
                    PISOVSTWO (4) BYTE DATA(0DBH, 0FH, 0C9H, 3FH),
                    THREEPISOV$2 (4) BYTE DATA(0E4H, 0CBH, 96H, 40H);
```

```
                 /***  CONVERSION OF BRG TO RADIAN    ***/

 119   2               CALL FMUL(.FRBR, .DEG$TO$RAD, .FBRG);

                 /***   TRUE TARGET BEARING     B = BR + COWN      ***/

 120   2           CALL FADD(.FBRG, .FCO, .FB);
 121   2           IF FCMPR(.FB, .TWO$PI, .CHECK1) THEN
 122   2              CALL FSUB(.FB, .TWO$PI, .FB);

                 /***   TARGET ANGLE   A = B + 180 - CT    ***/

 123   2           CALL FADD(.FB, .PI$FLOAT, .F$A);
 124   2           CALL FSUB(.F$A, .FCT, .F$A);
 125   2           IF FCMPR(.F$A, .TWO$PI, .CHECK1) THEN
 126   2             CALL FSUB(.F$A, .TWO$PI, .F$A);

 127   2         END SW$GEN$TRG$VALI;

                 /*************************************************************
                  ****
                 *
                 *   SW$GEN$TRG$VAL$II:
                 *
                 *   THIS PROCEDURE IS USED TO CALCULATE THE FOLLOWING PARAME
                  ERS.
                 *
                 *   FXO - CROSS COMPONENT OF OWN-SHIP VELOCITY
                 *   FYO - LINE
                 *   FYT -   "           "     "  TARGET      "
                 *   FXT - CROSS         "     "  OWN-SHIP    "
                 *
                 ***************************************************************
              -   ****/
 128    1        SW$GEN$TRG$VAL$II: PROCEDURE;
 129    2          CALL COS$SIN(.FBRG, .COS$FBRG, .SIN$FBRG);
 130    2          CALL FMUL(.COS$FBRG, .FSO, .FYO);
 131    2          CALL FMUL(.SIN$FBRG, .FSO, .FXO);
 132    2          CALL COS$SIN(.F$A, .COS$F$A, .SIN$F$A);
 133    2          CALL FMUL(.COS$F$A,.FST, .FYT);
 134    2          CALL FMUL(.SIN$F$A, .FST, .FXT);
 135    2        END SW$GEN$TRG$VAL$II;

                 /**************************************************************
              -   *****
                 *
                 *   SW$GEN$TRG$VAL$III:
                 *
                 *    THIS PROCEDURE IS USED TO CALCULATE THE FOLLOWING PARAM
              -   TERS
                 *
                 *    PARAMETERS:
                 *
                 *    FDR - RANGE RATE
                 *    FRLBR - LINEAR-BEARING RATE
                 *    FDBR - RELATIVE ANGULAR BEARING RATE
                 *
                 *    DR = 2.563(YO + YT)
                 *    FDBR = XO + XT
```

```
                  *     DBR = 1936*RDBR/R
                  *
                  *
                  ****************************************************************
                   *****/
136    1          SW$GEN$TRG$VAL$III: PROCEDURE;
137    2          DCL FP$MIN$1 (4) BYTE DATA (00H, 00H, 80H, 0BFH);
138    2          DCL FP$0$563 (4) BYTE DATA (0C5H, 20H, 10H, 3FH),
                        FP$1936 (4) BYTE DATA (00H, 00H, 0F2H, 44H);

139    2          CALL FMUL(.FGRNG, .FP$1, .FRNG);
140    2          CALL FMUL(.FGELV, .FP$1, .FELV);

                  /***     FDR = - 0.563( FYO + FYT )    ***/
141    2          CALL FADD(.FYO, .FYT, .FDR);
142    2          CALL FMUL(.FDR, .FP$MIN$1, .FDR);
143    2          CALL FMUL(.FDR, .FP$0$563, .FDR);

                  /***    FRDBR = FXO + FXT      ***/

144    2          CALL FADD(.FXO, .FXT, .FRDBR);
                  /***     FDBR = 1936*RDBR / R    ***/

145    2          CALL FDIV(.FRDBR, .FRNG, .FDBR);
146    2          CALL FMUL(.FDBR, .FP$1936, .FDBR);
147    2          END SW$GEN$TRG$VAL$III;

                  /***************************************************************
                   **
                   *
                  *   SW$GEN$TRG$VAL$IV:
                  *
                        THIS PROCEDURE IS USED TO CALCULATE THE FOLLOWING PARAM
              -    ETERS
                  *
                  *    PARAMETERS:
                  *
                  *    CR    - GENERATED PRESENT RANGE
                  *    CBR   - GENERATED RELATIVE TARGET BEARING
                  *
                  ***********************************************************
              -    **/
 148   1          SW$GEN$TRG$VAL$IV: PROCEDURE;
 149   2          DCL FDT$DR (4) BYTE,
                      FDT$DBR (4) BYTE;
 150   2          DCL FDT (4) BYTE DATA (00H, 00H, 80H, 3FH);

 151   2          CALL FMUL(.FDT, .FDR, .FDT$DR);
 152   2          CALL FMUL(.FDT, .FDBR, .FDT$DBR);

                  /*** CRNG = FRNG + FDT*FDR    ***/

 153   2          CALL FADD(.FRNG, .FDT$DR, .FCR);

                  /***    CBR = BRG + DT*DBR     ***/

 154   2          CALL FADD(.FBRG, .FDT$DBR, .FCBR);

 155   2          END SW$GEN$TRG$VAL$IV;
```

```
                  /*****************************************************
                  *
                  *  PREDICTED TARGET VALUES:
                  *
                  *   THIS PROCEDURE IS USED TO CALCULATE THE PREDICTED
                  *   TARGET RANGE AND BEARING .
                  *
                  *   PARAMETERS:
                  *
                  *   PRNG - PREDICTED TARGET RANGE
                  *   PBRG - PREDICTED BEARING
                  *
                  *****************************************************/
 156   1          SW$PRED$TRG$VAL: PROCEDURE;

 157   2          DCL TF$DR (4) BYTE,
                      TF$DBR (4) BYTE;

                  /***   PREDICTED RANGE   ***/

 158   2             CALL FMUL(.TF, .FDR, .TF$DR);
 159   2             CALL FADD(.FCR, .TF$DR, .PRNG);

                  /***   PREDICTED BEARING   ***/

 160   2             CALL FMUL(.TF, .FDBR, .TF$DBR);
 161   2             CALL FADD(.FCBR, .TF$DBR, .PBRG);

                  /*  FOR REAL APPLICATIONS AS DEPEND ON TYPE OF GUN RANGE
                      TABLE MUST BE INSERT TO THE PROGRAM THAT INCLUDES
```

AD-A102 693   NAVAL POSTGRADUATE SCHOOL MONTEREY CA   F/G 9/2
MICROCOMPUTER BASED SHIP-BOARD GUN CONTROL SYSTEM.(U)
MAR 81 A ERDOGAN

UNCLASSIFIED   NL

7 OF 7
AD
A102693

END
DATE
FILMED
9 81
DTIC

```
                    RANGE, TIME OF FLIGHT, AND ALL CORRECTIONS' FACTORS */

 152   2       END SW$PRED$TRG$VAL;

 163   1       CALL INIT$FP;

 164   1          DO WHILE 1;
 165   2            CALL SW$GEN$TRG$VALI;

 166   2            CALL SW$GEN$TRG$VAL$II;

 167   2            CALL SW$GEN$TRG$VAL$III;

 168   2            CALL SW$GEN$TRG$VAL$IV;

 169   2            CALL SW$PRED$TRG$VAL;

 170   2            CALL FLOAT$TO$BINARY(.PBRG(0), .BIN$BRG);

 171   2            CALL FLOAT$TO$BINARY(.PELV(0), .BIN$ELV);

 172   2          END;

 173   1       CALL EXIT;

 174   1       END SW$3FCS;
```

LIST OF REFERENCES

1. Babin, O.P. and Seaman, R.S., A Microcomputer Based Plasma Display System, M.S. Thesis, Naval Postgraduate School, Monterey, 1978.

2. Goncalves, A.L.S. and Bravo, J.E.D., A Microcomputer Based Shipboard Surface-Subsurface Contact Plotter System, M.S. Thesis, Naval Postgraduate School, Monterey, 1978.

3. Mariatequi, J.F.C. and Hall, Jr., I.N., Microcomputer Based Interactive Display System, M.S. Thesis, Naval Postgraduate School, Monterey, 1979.

4. Hastings, C. Jr., Approximations for Digital Computers, Princeton, Princeton University Press, New Jersey, 1955.

5. ISIS-II PL/M-80 Compiler Operator's Manual, Intel Corporation, Santa Clara, California 1977.

6. ISIS-II System User's Guide, Intel Corporation, Santa Clara, California, 1978.

7. 8080/8085 Assembly Language Programming Manual, Intel Corporation, Santa Clara, California, 1978.

8. PL/M 80 Programming Manual, Intel Corporation, Santa Clara, California, 1976.

9. McCracken, D.D., A Guide to PL/M Programming for Microcomputer Applications, Addison Weley, 1978.

10. Intellec Microcomputer Development System Hardware Reference Manual, Intel Corporation, Santa Clara, 1978.

11. Slaughter, J.B. and Lackowski, D.H., Digital Control of a Tactical Weapon with a Shipboard G-P Digital Computer, paper presented at National Convention on Military Electronics, 7th PROCEEDINGS, IEEE, San Diego, 1963.

12. Modular Servo System MS150, DC, Synchro And AC, Advanced Experiments Manual, V. 3, pp. 28-29, Feedback Instruments, 1980.

13. Kerns, K.H. and Cooper, R.S., A Microcomputer Solution to Maneuvering Board Problems, M.S. Thesis, Naval Postgraduate School, Monterey, 1973.

14. Scheid, F., Theory and Problems of Numerical Analysis, Schaum's Outline Series, McGraw Hill, 1968.

15. Plasma Display Set Technical Manual, Vols. I and II, Science Applications Inc., 1976.

16. SBC 310, High-Speed Mathematics Unit Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1977.

17. Ogata, K., Modern Control Engineering, Prentice-Hall, 1970.

18. Newman, W.M. and Sproull, R.F., Principles of Interactive Computer Graphics, McGraw-Hill Computer Science Series, 1973.

19. Klingman, E.E., Microprocessor Systems Design, Prentice-Hall, 1977.

20. Peatman, J.B., Microcomputer-Based Design, McGraw-Hill, 1977.

21. Wrigley, W. and Hovorka, J., Fire Control Principles, McGraw-Hill, 1959.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	2
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
5.	Associate Professor U.R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
6.	Professor R. Panholzer, Code 62Pz Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
7.	Adjunct Professor R. Baird, Code 62Bd Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
8.	T.C. Deniz Kuvvetleri Komutanligi Egitim Sube Bahanliklar-ANKARA, TURKEY	5
9.	O.D.T.U. Kutuphanesi ANKARA, TURKEY	1
10.	Bogazici Universitesi Kutuphanesi, ISTANBUL-TURKEY	1
11.	I.T.U. Kutuphanesi ISTANBUL-TURKEY	1
12.	Dz. Yzb. Ahmet Endoğan Pembe Gūl Sok. No: 17/5 Suadiye-ISTANBUL-TURKEY	2